UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

A FAST PROCEDURE FOR GENERATING RANDOM NUMBERS BY A MODIFICATION OF THE MARSAGLIA-MACLAREN METHOD

by

IOANNIS ELIAS IOANNOU, DIPLOMA IN MATHEMATICS

FROM THE UNIVERSITY OF ATHENS

A Thesis

Prepared under the Supervision of

Professor G.B.Brundrit

In Fulfilment of the Requirements for the

DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

Cape Town

May, 1974

ACKNOWLEDGEMENTS

My first thanks are to my supervisor, G.B. Brundrit, Professor of Applied Mathematics at the University of Cape Town, for his advice and for the time he devoted in reading this report.

My thanks are extended to the Computer operators, Fatima Allie, C.H. Harris, Ghani Richards and Neville Alcock who helped me so much while developing my programs.

To my friends, architect, Bill Violijis and Alekos Demopoulos who drew the flowcharts of this report, I express my deepest gratitude.

I should also like to express my appreciation to my friends, C. Goldberg and M. Hind who made valuable suggestions which I was able to use.

My thanks are due to Mrs. S. Drew and Mrs. P. Alexander for the trouble they took over typing the first draft of this thesis.

Finally, I should like to thank my Aunt Koula and my Uncle Paul Ioannou, without whose financial assistance and interest in my having a higher education, I should never have reached this stage in my studies.

LIST OF TABLES

ABSTRACT

Marsaglia and Maclaren combined two linear congruential generators in order to produce a pseudo random number sequence uniformly distributed in the range $[0,2^{35})$. Their method is a considerable improvement compared with the primitive linear congruential method at the cost of greater generation time. In this thesis, a simple modification of the Marsaglia-Maclaren method is presented in which there is an alleviation of the increased generation time, and a slight further increase in randomness. The modified generator is tested extensively in a variety of statistical tests and simulation problems.

CONTENTS

INTRODUCTION

A simulation based on random behavior apparently requires a mechanism for generating sequences of events, where each sequence obeys a probability law governing a particular component of the random behavior in question. In practice, many systems include activities that behave randomly, in which case the exact sequence of events is not known.

Sometimes, the activity is not intrinsically random in nature, but complete information about it's behavior is not known and it should be described as a random activity. Possibly, complete information about its behavior is available or it can be derived mathematically, but it is so complex that it is more convenient to describe the activity as being random.

A variable representing the outcome of a random activity is said to be a stochastic variable. Although the exact sequence of values taken by a stochastic variable is not known, the range of values over which it can vary, and the probability with which it will take the values may be known or assumed to be known. Stochastic variables are therefore discussed in terms of functions which describe the probability of the variable taking various values.

Numbers which can be produced "randomly" by a certain procedure are useful in a wide variety of applications. For example, simulation, sampling, Numerical Analysis, computer programming, Decision making etc. Actually we cannot speak about a random number. Rather, we speak of a sequence of independent random numbers with a specified distribution. D.H. Lehmer (1951) gave the following definition of a random sequence of numbers.

"A random sequence is a vague notion embodying the idea of a sequence in which each term is unpredictable to the unitiated and whose digits pass a

certain number of tests, traditional with statisticians and depending somewhat on the uses to which the sequence is to be put." At first people who needed random numbers in their scientific work would draw balls out of a well-stirred urn or would roll dice or deal out cards. Shortly after computers were introduced, people began to search for more efficient ways to obtain random numbers in computer programs.

John von Neumman first suggested using the arithmetic operations of a computer. His "middle-square" method is well-known. His idea was to take the square of the previous random number and to extract the middle digits. This procedure can be repeated in order to get a sequence of random numbers. This method has proved to be [11] a very poor source of random numbers, because the period of the sequence seems to be very short.

D.H. Lehmer introduced an analytical method to produce sequences of random numbers. This is the well-known Linear Congruential Method. The desired random-sequence is obtained by the equation $X_{n+1} = (aX_n + c) \bmod m$, where X is an integer, a the multiplier, c the increment and m the modulus. We shall examine this method in more detail in Chapter one.

Other analytical methods which have been proposed are the quadratic congruential method [11], the Fibonacci sequence and the additive number generator [11]. But neither of the above methods can be compared with the linear congruential method [11].

Marsaglia-Maclaren investigated [12] the possibility of using a table of random numbers. This turns out to be practical on a large computer with buffered input. The table of random numbers is an obvious alternative to the use of numerical procedure for generating random numbers.

The above authors observed [12] that the linear congruential generator is unsuitable for high resolution Monte-Carlo problems. They devised a new numerical method of generating random numbers. They simply

combined two linear congruential generators and they found out that the new generator has much better statistical properties than the simple linear congruential generator. The algorithm which was given by them is briefly described below. (Detailed description of this algorithm is given in Chapter 2).

Consider two linear congruential generators

(1) $X_{k+1} = (aX_k + c) \bmod 2^{35}$

(2) $Y_{k+1} = (a_1 Y_k + c_1) \bmod 2^{35}$

A table of $128 = 2^7$ locations in core is filled with the numbers $X_1, X_2, \ldots, X_{128}$. Then to generate U_k the kth random number to be used, the first 7 bits of Y_k are used as an index to retrieve U_k from the table. The location of U_k in the table is refilled with the next number from the generator (1). We observe that the values from the first generator are re-ordered randomly so as to form the output sequence. The second generator controls the re-ordering of the sequence.

This method has the disadvantage of being slower than the linear congruential method. The time required to generate this sequence is twice as long as it takes to generate the sequence X_k alone. We attempt in this report to alleviate the time problem by modifying slightly the original Marsaglia-Maclaren method.

Instead of taking two linear congruential generators, only one is taken which is re-ordered randomly so as to form the output sequence. The same generator controls the re-ordering of the sequence. The CPU time required to output a random number with the second method is reduced by 30-35 percent. A comparison is made between the two methods with a variety of statistical tests and from the results is noted that the modified method has better randomness than the Marsaglia-Maclaren method.

Many standard results are quoted in the thesis and these are taken mainly from the book, "The art of computer programming", volume 2 semi-numerical algorithm by D.E. Knuth.

The whole thesis is divided basically in seven chapters. In Chapter I the proper choice of the parameters of the linear congruential generator is examined. This will help us to choose properly the parameters of the auxiliary generator of the modified Marsaglia-Maclaren method. In Chapter 2 the original and the modified methods are presented. In Chapter 3 a comparison is made between the two methods with the best-known statistical tests. Chapters 4 and 5 test the modified generator in a variety of simulation problems. In Chapter 6 other types of random quantities following discrete and continuous probability distributions are generated. In Chapter 7 some interesting queuing problems are solved using as a source of random numbers some generators of the preceding chapter.

CHAPTER I

RANDOM NUMBER GENERATION

1.1 INTRODUCTION.

In this chapter we shall be concerned with random sequences of numbers and attempts to specify suitable algorithms for generating such sequences.

By random sequence of numbers we shall mean a sequence in which each member is obtained by chance, is independent of any other member, but has a specified probability of falling in any given range of values.

A uniform distribution is one in which each possible number is equally probable, and every distribution in this report is understood to be uniform unless some other distribution is specified.

Many simulation problems and other fields of application require the generation of a random sequence of real numbers {Vn}, uniformly distributed between 0 and 1. One convenient way to do this is to generate a random sequence of integers {Xn} uniformly distributed between 0 and m and use the quotient Vn=Xn/m.

The end point m would have to be a very large integer. The main problem is to ensure that any sequence of integers generated by an algorithm is suitably random.

We shall be concerned with sequences generated by the linear congruential generator and various similar algorithms such as the Marsaglia-MacLaren generator.

In this chapter we shall investigate carefully the choices of values for the parameters in these algorithms which are most appropriate for generating random sequences.

I.2 LINEAR CONGRUENTIAL GENERATOR.

Since the preceding section has emphasized the desirability of independence and uniformity, it may be surprising to find that the most commonly employed random number generation algorithm produces a nonrandom sequence of numbers, each number being completely determined by its predecessor and, consequently, all numbers being determined by the initial number, but provided the parameters of the generator are carefully chosen the numbers appear to be sufficiently independent and uniform for many practical purposes. To emphasize their inherently nonrandom character, we call these numbers pseudorandom numbers.

This generator, the linear congruential generator, has the form:

(1) $X_{n+1}=(\alpha X_n+c) \bmod m \quad n\geq o$

where X_o = starting value , $X_o \geq o$

α = multiplier $\alpha > o$

c = increment $c \geq o$

m = modulus $m > X_o$, $m > \alpha$, $m > c$

When the increment is taken to be zero, the generator is called multiplicative congruential generator and when $c \neq o$, mixed congruential generator.

Let us consider a specific example.

If $\alpha=3$, $m=5$, $c=1$, $X_o=1$ the produced sequence will be,

4,3,0,5,1,4,3,0,5,1,4, ...

We see that after the first 5 terms which are different, the sequence is repeated ad infinitum.

The length of the sequence until a number is repeated is called the period length, and clearly the period is less than or equal to the modulus m. In the example, the period is 5. An equivalent formulation of the l.c.g. is:

(2) $X_{n+k}=(\alpha^k X_n+(\alpha^k-1)C/(\alpha-1))\text{mod}m,\ k\geq o,\ n\geq o$, which expresses the (n+k)th term directly in terms of the nth term. This can be proved by induction on k.

Generally it is highly desirable that our generator has the longest possible period.

As we shall see later in this chapter the restriction c=o cuts down the period length so Thomson [Comp. J. I (1958), 83, 86] proposed the idea of taking $c\neq o$ in order to obtain longer periods.

In this report we shall choose the modulus m to be equal to computer size word.

We make this choice for two reasons.

First, if the choice is the computer size word, the longest possible period can be attained and secondly the computation of the expression $(\alpha X+c)\text{mod}m$ will be fast.

Univac's 1108 word consists of 36 bits, the highest order bit denoting the sign of the number.

So the largest representable positive integer in the machine is the number $2^{35}-1$. If we wish the whole set of integers $(0,1,2,\ldots,2^{35}-1)$ to be produced by the generator (1) we should take the modulus m to be equal to 2^{35}.

1.3 CHOICE OF THE MULTIPLIER A.

A long period is essential for any sequence which is to be used as a source of random numbers, in a simulation problem.

In this section we shall show how to choose the multiplier α and the increment c, as to give the maximum period length, the modulus m.

The following theorem tells us what the appropriate choice of α and c should be:

Theorem A.

The mixed linear congruential sequence has a period of length m iff:

(i) c is relatively prime to m.

(ii) $b = \alpha - 1$ is a multiple of P, for every prime P dividing m.

(iii) b is a multiple of 4 if m is a multiple of 4.

The sufficiency of the conditions (i), (ii), (iii) was shown by Hull and Dobel (Siam review 4 (1962) 230-254).

We shall restate the theorem to fit to our needs.

We observe that the modulus $m=2^{35}$ has only one prime factor, the number 2.

The theorem becomes:

Theorem B.

The linear congruential sequence has a period of length $m=2^{35}$ iff:

(i) c is odd.

(ii) $b = \alpha - 1 = 0 \bmod (4)$.

We now turn to a consideration of the parameters which give maximum period length.

Let us assume a multiplier of the form $\alpha = 2^k + 1 \quad 2 \leq k < 35$ (I). Apparently multipliers of the form in (I) satisfy the maximum period length criterion when the modulus is a power of two. However, some five years of experimentation ref.(II) with this method show that multipliers having the form in (I) must be avoided.

The main reason for this is the "low potency" of the multiplier 2^k+1.

We give the definition of the term potency.

The potency of a linear congruential sequence with maximum period is defined to be the least integer s such that:

(2) $b^s \equiv 0 \bmod m$ where $b = \alpha - 1$

Since b is a power of 2 and $m = 2^{35}$ such an integer will always exist.

Let us consider now the linear congruential generator.

(3) $X_{n+1} = (\alpha X_n + c) \bmod 2^{35}$.

Since 0 occurs somewhere in the period, we may take Xo=0, so

$X_1, X_2 \ldots . Xn$ become,

$X_1 = c \bmod 2^{35}$

$X_2 = (\alpha c + c) \bmod 2^{35}$

$X_3 = \alpha^2 c + \alpha c + c = \frac{\alpha^3 - 1}{\alpha - 1} c = \frac{\alpha^3 - 1}{b} c \bmod 2^{35}$

$\vdots$

$X_n = \frac{\alpha^n - 1}{b} c \bmod 2^{35}$

and if we expand $\alpha^n - 1 = (b+1)^n - 1$ by the binomial theorem, we shall have,

$$(4)\quad X_n = c\left[n + \binom{n}{2}b + \ldots\ldots + \binom{n}{s}b^{s-1}\right] \bmod 2^{35} \quad .$$

All terms b^s, b^{s+1}, ... in (4) will be zero, since they are multiples of 2^{35}.

If the potency is 1 (for example if we take α to be 1), we have $X_n = nc \bmod 2^{35}$.
This sequence has too simple a pattern to be considered a random sequence.

If potency is 2 we have from (4) $X_n = cn + cb\binom{n}{2}$ and surely this cannot be considered random, as the difference between two successive terms of the sequence has a very simple form,

$$X_{n+1} - X_n = c + cbn.$$

Reasonable results have been reported [11] when potency is five or more. So the multiplier should have potency at least five in order to get a reasonably random seqeuence.

We now turn our attention to the multipliers of the form 2^k+1, $2 \leq k < 35$. If $k=2, \ldots, 8$ the potency varies from 18 to 5 and although these multipliers are acceptable from the standpoint of potency, should be eventually rejected as we shall see later since small multipliers do not contribute to the randomness of the linear congruential generator. As k takes on the values 9,10, ..., 34 the size of the multiplier increases but the potency decreases so these multipliers are not acceptable from the standpoint of potency. We have thereby rejected all multipliers of the form 2^k+1 when $m = 2^{35}$.

When a multiplier has the form in (1) it is relatively easy to find its potency. Suppose we are given the multiplier

A=3141392653, how shall we find its potency?

The following theorem gives the answer.

Theorem C.

- Given that $m=P_1^{e_1}.P_2^{e_2}\ldots P_l^{e_t}$ and $\alpha=l+kP_l^{f_l}\ldots P_t^{f_t}$ where α satisfies the conditions of the theorem A and k is relatively prime to m, the potency of α is $\max(\lceil e_1/f_1\rceil, .., \lceil e_t/f_t\rceil)$.

$P_1, P_2, .. P_k$ are prime numbers. With $\lceil x\rceil$ (ceiling of x) we denote the least integer greater than or equal to x.

Proof.

Let $\frac{e_\lambda}{f_\lambda}$ be the maximum, then $s=\lceil\frac{e_\lambda}{f_\lambda}\rceil$ and $s\geq\lceil\frac{e_1}{f_1}\rceil$ $s\geq\lceil\frac{e_2}{f_2}\rceil,\ldots.$ $s\geq\lceil\frac{e_t^\lambda}{f_t}\rceil$ or $s\geq\frac{e_\lambda}{f_\lambda}$, $s\geq\frac{e_1}{f_1}$, ... $s\geq\frac{e_t}{f_t}$ then

$sf_1\geq e_1$

$sf_2\geq e_2$

.....

$sf_t\geq e_t$

or

$sf_1 = e_1+k_1$

$sf_2 = e_2+k_2$ where $k_1, k_2, \ldots k_4$ are integers.

$sf_t = e_t+k_t$

so

$b^s=k^sP_1^{f_1s}P_2^{f_2s}\ldots P_t^{f_ts}=k^sP_i^{e_1+k_1}\ldots P_t^{e_t+k_t}=$

$=k^sP_1^{k_1}P_2^{k_2}\ldots P_t^{k_t}(P_1^{e_1}\ldots P_t^{e_t})=0 \bmod m.$

The calculation becomes easier if the modulus is 2^{35}, since the only prime factor of 2^{35} is 2.

Now we can easily find the potency of $\alpha=3141592653$.

We have $b=\alpha-1=3141592652=2^2 \times 785398163$.

Since 785398163 is relatively prime to $m=2^{35}$ the potency is $\left\lceil \frac{35}{2} \right\rceil = 18$.

We now prove the following important theorem.

Theorem D.

If $m=2^e \geq 8$ the maximum potency is achieved when $\alpha=5 \bmod 8$.

Proof.

If we consider a generator having full period, then

(1) $b \equiv 0 \bmod (4)$ Since $b=\alpha-1$ we have

(2) $\alpha \equiv 1 \bmod (4)$ or $\alpha=1+4k$, if now $k=2n \rightarrow$

(3) $\alpha=1 \bmod 8$ and if $k=2n+1 \rightarrow \alpha=5 \bmod 8$ (4).

From (3) since $b=\alpha-1$ we have $b=0 \bmod 8$ and from (4) $b=4 \bmod 8$.

So when $\alpha=1 \bmod 8 \rightarrow b=0 \bmod 8$ (b is a multiple of 8) and when $\alpha=5 \bmod 8 \rightarrow b=4 \bmod 8$ or $b=4+8k=4+4k.2 = +4(1+2k)$ or b is an odd multiple of 4.

If now b is an odd multiple of 4 (that is $\alpha \equiv 5 \bmod 8$, and b1 is a multiple of 8, that is $\alpha \equiv 1 \bmod 8$, we shall show that b has greater potency than b1.

Actually $b=k4=k.2^2$ where k odd, so $(k.2^2)^s=0 \bmod 2^{35}$ and the potency is $s=\left\lceil \frac{35}{2} \right\rceil=18$, while if $b1=k.8=k.2^3$, b1 has at most potency 12.

We see now that in order to have the maximum period and the highest potency, we must choose the multiplier α, such that $\alpha \equiv 5 \bmod 8$.

Actually the concept of potency is only one criterion for the choice of the multiplier. In the next section we shall discuss the

"spectral test" for multipliers of linear congruential sequences. This is an important criterion which includes potency and the size of multiplier as special cases.

1.4 SPECTRAL TEST.

As we have seen the multiplier α should be chosen in such a way that it satisfies the condition $\alpha \equiv 5 \bmod 8$. It has been proved by RR Coveyou an R.D. MacPherson (2) that if a random number generator is to be used extensively, the multiplier α should also be chosen so that it passes their "spectral test".

This test has been developed to explain the statistical behaviour of K-tuples of random numbers from congruential generators.

This test eventually examines the effectiveness of a multiplier α in a linear congruential sequence. The modulus m of the congruential generator is taken to be very large, usually the computer size word. Let us consider now the spectral test from a mathematical point of view.

Given that $F(t_1, .., t_n)$ is any complex-valued function defined for all combinations of integers t_k, where $0 \leq t_k < m$ for $1 \leq k \leq n$, define the Fourier transform of F by the following rule.

$$(1)\quad f(S_1, S_2, \ldots, S_n) = \sum_{0 \leq t_1, t_2, \ldots, t_n < m} \exp\left(-\frac{2\pi i}{m}(S_1 t_1 + .. + S_n t_n)\right) F(t_1, .. t_n)$$

This function f is defined for all combinations of integers S_k; it is periodic, in the sense that,

$f(S_1, .., S_n) = f(S_1 \bmod m, .., S_n \bmod m)$.

In fact if we take the expression $\exp(\frac{-2\pi i}{m}(S_1 t_1 + .. + S_n t_n))$

and substitute S_1 by $S_1 \bmod m = S_1 + k_1 m$

S_2 by $S_2 \bmod m = S_2 + k_2 m$

.....

$$S_n \text{ by } S_n \bmod m = S_n + k_n m$$

we shall have $\exp(-\frac{2\pi i}{m}(S_1t_1+\ldots+S_nt_n)) =$

$= \exp(-\frac{2\pi i}{m}(S_1t_1+\ldots+S_nt_n) - 2\pi i(k_1t_1+\ldots+k_nt_n)) =$

$= \exp(-\frac{2\pi i}{m}(S_1t_1+\ldots+S_nt_n))$ we arrive at (1).

In fact $e^{-2\pi i(k_1t_1+\ldots+k_nt_n)}=1$ if we take into consideration Euler's formula $e^{i\theta}= \cos\theta+i\sin\theta$.

We shall prove now that the original function $F(t_1,..,t_n)$ can be reconstructed by its transform as follows.

(2) $$F(t_1,\ldots,t_n) = \frac{1}{m^n} \sum_{0\le S_1,..,S_n<m} \exp(\frac{2\pi i}{m}(S_1t_1+..+S_nt_n)f(S_1,..,S_n)$$

Taking into account formula (1) we have,

$$F(t_1,..,t_n)=\frac{1}{m^n} \sum_{0\le S_1,..,S_n<m} \sum_{0\le u_1,..,u_n<m} \exp(\frac{2\pi i}{m}(S_1t_1+..+S_nt_n)\exp(-\frac{2\pi i}{m}(S_1u_1+$$

$$+..+S_nu_n),$$

$$F(t_1,..,t_n)=\frac{1}{m^n} \Sigma\Sigma \exp(\frac{2\pi i}{m}(S_1(t_1-u_1)+..+S_n(t_n-u_n)) \to$$

$$F(t_1,..,t_n)=\frac{1}{m^n} \sum_{0\le u_1,..,u_n<m} F(u_1,..,u_n)(\sum_{0\le S_1<m} w^{S_1(t_1-u_1)})\ldots(\sum_{0\le S_n<m} w^{S_n(t_n-u_n)})$$

where w is the m^{th} root of unity $e^{\frac{2\pi i}{m}}$

Let us calculate now one of the inner sums.

$\sum_{0\le S_\lambda<m} w^{S_\lambda(t_\lambda-u_\lambda)} = \frac{w^{m(t_\lambda-u_\lambda)}-1}{w^{t_\lambda-u_\lambda-1}}$ because the sum is progressive series.

If now $t_\lambda-u_\lambda \neq 0 \bmod m$ we have $\Sigma w^{S_\lambda(t_\lambda-u_\lambda)} = 0$ and if $t_\lambda-u_\lambda = 0 \bmod m$ we have $\Sigma w^{S_\lambda(t_\lambda-u_\lambda)} = 1$, taking again into account Euler's formula.

So the above formula becomes

$$F(t_1,..,t_n)=\frac{1}{m^n}\sum_{0\le u_1,..,u_n<m} F(u_1,..,u_n)=\frac{1}{m^n}\ m^n\ F(t_1,..,t_n) =$$

$F(t_1,..,t_n)$, since we can select n values $u_1,..u_n$ out of m in m^n ways.

The formula (2) can be expanded into sines and cosines, so it resembles an infinite Fourier series.

The value $(1/m^n)f(S_1,..,S_n)$ represents the amplitude of an n-dimensional complex plane wave with frequences $S_1/m,..,S_n/m$ if $F(t_1,..t_n)$ is written as a superposition of such waves.

As a consequence of relations (1) and (2), it is possible in theory to determine any property of F from its transform f and conversely.

Suppose now that Xo,X1,.., is a sequence of integers with $0\le X_k<m$ and let n be a small positive integer. We define now,

$$(3)\quad F(t_1,..,t_n)=\lim_{N\to\infty}\frac{1}{N}\sum_{0\le k<N}\delta X_{k+1}\ \delta X_{k+1t_2}\ ..\delta X_{k+n-1t_n}\quad ,$$

that is $F(t_1,..,t_n)$ is the limiting density of the number of appearances of the n-tuple $(t_1,..,t_n)$ as n consecutive elements of the sequence $X_o, X_1,...$

Since all sequences $X_o,X_1,..$ are periodic, we may assume the limit in (3) exists and we may set N equal to the period length.

The value now of the expression $F(t_1,..,t_n)$ should be equal to $1/m^n$, since in a truly random sequence for the uniform distribution the probability for each possible n-tuple to appear is $1/m^n$.

The Fourier transform of (3) has now the form,

$$(4)\quad f(S_1,..,S_n)=\lim_{N\to\infty}\frac{1}{N}\sum_{0\le k<n}\exp(-\frac{2\pi i}{m}(S_1X_k+..+S_nX_{k+n-1}))$$

In a truly random sequence, this should be the transform of the constant function $1/m^n$.

We try now to find the value of the expression $f(S_1,..,S_n)$ provided that the sequence $X_o,X_1,..,$ is a truly random sequence.

If $S_1=S_2=\ldots=S_n=0$ modm from (4) we have,

$$f(S_1,..,S_n)=\lim_{N\to\infty}\frac{1}{N}\sum_{0\le k<N}\exp(-\frac{2\pi i}{m}(\lambda_1 X_k+\ldots+\lambda_n X_{k+n-1})m)$$

$$=\lim_{N\to\infty}\frac{1}{N}(\Sigma\exp(-2\pi i(\lambda_1 X_k+..+\lambda_n X_{k+n-1}))$$ and taking into consideration Euler's formula, the above equation becomes,

$f(S_1,..,S_n)=\lim_{N\to\infty}\frac{1}{N}\sum_{0\le k<N}1)=N/N=1$, other wise we shall have

$f(s_1,..,S_n)=0.$

So the following relations are derived,

$$(5)\quad f(S_1,..,S_n)=\begin{cases}1 \text{ if } S_1=S_2=..=S_n=0 \text{ modm}\\ 0 \text{ otherwise}\end{cases}$$

Any theoretical test, which finds the average of some function depending only on n consecutive values of the sequence, can, in principle be completely determined from the values $F(t_1,..,t_n)$.

For example, the probability that $X_k<X_{k+1}$ is the quantity $F(t_1,t_2)$ summed over the values $0\le t_1<t_2<m$.

Similarly, any theoretical test can in principle be determined from the transformed function $f(S_1,..S_n)$ in (4), since this function carries the same information as $F(t_n,..,t_n)$ does.

Therefore the deviation of $f(S_1,..,S_n)$ from the values (5) corresponding to a truly random sequence will be a useful test for randomness.

We shall consider now the case of the linear congruential sequence, defined by a=multiplier, m=modulus, c=increment and X_o=initial value, having the maximum period in accordance with the theorem A page 5.

For this sequence the transformation function is,

$$(6)\quad f(S_1,..,S_n)=\frac{1}{m}\sum_{0\le k<m}\exp(-\frac{2\pi i}{m}(S_1X_{k+}\ldots+S_nX_{k+n-1}))$$

We put N equal to the period length m.

In linear congruential generator the following formula holds,

(7) $X_{k+r}=a^r X_k+\frac{a^r-1}{a-1}c \bmod(m)$ (See page 4).

Now formula (6) becomes taking into account formula (7),

$$f(S_1,..,S_n)=\frac{1}{m}\sum_{0\le k<m}\exp(-\frac{2\pi i}{m}(S_1X_k+S_2(aX_k+\frac{a-1}{a-1}c)+S_n(a^{n-1}X_k+\frac{a^{n-1}-1}{a-1}c))=$$

$$=\frac{1}{m}\sum_{0\le k<m}\exp(-\frac{2\pi i}{m}(S_1X_{k+}...+S_na^{n-1}X_k+\frac{S_2a+..+S_na^{n-1}-S_1...-S_n}{a-1}c))=$$

(8) $$=\frac{1}{m}\sum_{0\le k<m}\exp(-\frac{2\pi i}{m}(S(a)X_k+\frac{S(a)-S(1)}{a-1}c))$$

where

(9) $S(a)=S_1+S_2a+...+S_na^{n-1}$

We assume now that the sequence has maximum period, so that all values X_k occur. Therefore if we substitute X_k by k where $0\le k<m$ relation (8) becomes, $\frac{1}{m}\sum_{0\le k<m}\exp(-\frac{2\pi i}{m}(S(a)k+\frac{S(a)-S(1)}{a-1}c))$ or

$$f(S1,S2,..,Sn)=\frac{1}{m}\sum_{0\le k<m}\exp(-\frac{2\pi i}{m}(S(a)K)\exp(-\frac{2\pi i}{m}c\frac{(S(a)-S(1))}{a-1})$$

$$=\frac{1}{m}e^{-\frac{2\pi i}{m}c(\frac{S(a)-S(1)}{a-1})}\cdot\sum_{0\le k<m}\exp(-\frac{2\pi i}{m}S(a)k).$$

But the above sum is a geometric series whose sum is,

$\frac{\exp(-2\pi i\ S(a))-1}{\exp(-\frac{2\pi i}{m}\ S(a))-1}$. If $S(a)=0$ modm the geometric series have sum

equal to 1, if $S(a)\neq 0$ modm then their sum is zero.

So we have finally,

(10) $f(S_1,S_2,..,S_n)=\exp(\frac{-2\pi ic}{m}(\frac{S(a)-S(1)}{a-1}))\delta(\frac{S(a)}{m})$

where $\delta(\frac{S(a)}{m})=1$ if $S(a)=0$ modm, otherwise 0.

We can see from equation (10) that the value of c in a linear

congruential sequence of maximum period does not affect the Fourier coefficients except to change the argument of the complex number $f(S_1,S_2,..,S_n)$; in other words the absolute value of $f(S_1,..,S_n)$ is independent of c.

Now the natural question arises. Is it possible to choose c so that the nonrandom effect of one wave $f(S_1,..,S_n)$ might be cancelled out by an 'opposite' effect of another wave $f(S_1',..,S_n')$?

The answer is no. Since $F(t_1,..,t_n)$ is uniquely determined by $f(S_1,..,S_n)$ and conversely, there can be no genuine cancelling between waves with different wave length components.

But certain statistical quantities calculated from $f(S_1,..,S_n)$, such as the serial correlation coefficient can be significantly affected by the choice of c (see page 19).

Recall that the expression $f(S_1,..,S_n)/m^n$ represents the amplitude of a n-dimensional complex wave, and by convention this wave may be assigned a wave number ν corresponding to its frequency, where,

(11) $\nu = \sqrt{S_1^2+...+S_n^2}$ where $-\frac{m}{2} \leq S_k \leq \frac{m}{2}$ for $k=1,..,n-1$

The wavelength is given by the formula,

$$L=(S_1^2+S_2^2+...+S_n^2)^{-\frac{1}{2}}.$$

It has been shown [3] that such a wave is harmful if the wavelength is long, harmless if it is short.

In this regard it is clear that low-frequency components are more damaging to randomness than high-frequency are.

It has been proved [11] that if ν_n is the smallest non-zero value of the wave number (11) for which $f(S_1,..,S_n)\neq 0$ in a linear congruential sequence with maximum period, then the sequence $X_o/m, X_1/m, X_2/m,...$ represents a sequence of random numbers uniformly distributed between 0 and 1, having accuracy $1/\nu_n$ with respect to the independence of n

consecutive values of the sequence averaged over the entire period.

Equation (10) shows us which waves appear in the Fourier transform of $F(t_1,..,t_n)$. We see that $f(S_1,..,S_n)=0$ except when

(12) $$S_1+S_2\alpha+...+S_n\alpha^{n-1}=0 \bmod(m),$$

and in this case $|f(S_1,..,S_n)|=1$ since $f(S_1,..,S_n)=\exp(-\frac{2\pi i c}{m}(\frac{S(\alpha)-S(1)}{a-1}))$.

Therefore for linear congruential sequences of maximum period, the smallest non-zero wave number is given by,

(13) $$\nu_n = \min \sqrt{S_1^2+...+S_n^2}$$ subject to the condition (12).

$S_1,S_2,..,S_n$ must not be all zero, because the wavelength will be infinite.

The parametric and general solution of the system (12) and (13) is given by the equations.

$$S_o=(mX_1-\alpha X_2-\alpha^2X_3-...-\alpha^{n-1}Xn)$$

$$S_1=X_2$$

.....

$$S_n=X_n.$$

So $\nu_n^2=\min(mX_1-\alpha X_2-...-\alpha^{n-1}X_n)^2+X_2^2+...+X_n^2$

It seems appropriate to give, αS a measure of the randomness the volume of the ellipse in n-space defined by the relation,, $(X_1m-\alpha X_2-...-\alpha^{n-1}X_n)^2+X_2^2+...+X_n^2\leq\nu_n^2$ since the volume of this ellipse serves as an indication of the probability that integer points $(X_1,..,X_n)$ - corresponding to a solution of the above system - are in the ellipse.

It is proposed to calculate the quantities,

(14) $$C_n=\frac{\pi^{n/2}\nu_n^n}{(n/2)!\,m}$$

in order to rate the effectiveness of the multiplier α in a linear congruential sequence of maximum period.

In this formula $(\frac{n}{2})!$ is the gamma function,

$\Gamma(\frac{n}{2}+1)=(\frac{n}{2})!=(\frac{n}{2})(\frac{n}{2}-1)....\frac{1}{2}\sqrt{\pi}$ for n=odd and π=3.14159....

Large values of C_n, correspond to randomness, small values to non-randomness.

Experience shows, that a multiplier passes the spectral test if the values $C_2, C_3, C_4...$ are all ≥ 0.1.

One consequence of this discussion is that we can place upper limits on the possible randomness of any congruential sequence.

Hermite's theorem says that:

$\nu_n \leq \gamma_n m^{1/n}$ where γ_n takes the respective values,

$1, (4/3)^{1/4}, 2^{1/6}, 2^{1/4}, 2^{3/10}, (64/3)^{1/12}, 2^{3/7}, 2^{1/2}$ for n=1,2,3,4,5,6,7,8.

The significance of this theorem is that there is an absolute upper limit for ν_n^2 which says that any method of generation of pseudo-uniform random numbers of the type we were discussing cannot be made arbitrarily good (in the sense of uniformity of distribution of strings of output elements) by a favourable choice of parameters.

Since the spectral test involves formidable integer arithmetic calculations, for example the calculation of m^2-m computer size word - it was impossible to put it in action since high precision integer arithmetic is required, and Univac 1108 lacks this facility. In this report we present a table, page A3, which contains some of the known multipliers and their corresponding values of C_2, C_3, C_4.

1.5 CHOICE OF THE INCREMENT c.

The general discussion of the congruential method would be incomplete without reference to an important relation which gives us some advice about the proper choice of the increment c.

We have so far obtained no criterion for choosing c, except that it be relatively prime to m.

Consider the statistic,

$$(1)\quad C = \frac{n\sum_{j=0}^{n-1}(U_jV_j) - (\sum_{j=0}^{n-1}U_j)(\sum_{j=0}^{n-1}V_j)}{\sqrt{(n\Sigma U_j^2 - (\Sigma U_j)^2)(n\Sigma V_j^2 - (\Sigma V_j^2))}}$$

This is the correlation coefficient, which is a measure of the extent U and V are linearly related. A special case of the above is the serial correlation coefficient (C* say) which is given by

$$(2)\quad C^* = \frac{n(U_0U_1+U_1U_2+\ldots+U_{n-1}U_0) - (U_0+U_1+\ldots+U_{n-1})^2}{n(U_0^2+U_1^2+\ldots+U_{n-1}^2) - (U_0+U_1+\ldots+U_{n-1})^2};$$

where $V_j = U_{(j+1) \bmod n}$, i.e. U_j and $U_{(j+1) \bmod n}$ are nearby terms.

A correlation (or serial correlation) coefficient always lies between -1 and +1. A zero correlation coefficient implies that the variables are linearly independent of each other.

A correlation coefficient of $\pm$ 1 implies that the variables are linearly dependent.

In the case of the linear congruential generator we wish nearby terms U_j and $U_{(j+1) \bmod n}$ to be independent and therefore require (2) to be "small" or zero.

Greenberger [6] expressed the above formula for C in terms of the multiplier α and increment c of the linear congruential equation taken over the full period m. He gave the following approximate formula for the serial correlation:

$$(3) \qquad C \approx \frac{1}{\alpha}\left(1-6\frac{c}{m}+6\left(\frac{c}{m}\right)^2\right)$$

The above formula shows us that small values of α must be avoided, else the serial correlation will be high. If $\alpha=\sqrt{m}$ relation (3) takes its minimum value.

Experience shows [11] that the multiplier α must vary between $\sqrt{m}$ and $m-\sqrt{m}$. Similarly the above relation offers us an important criterion for choosing c.

If we choose c so that, in addition to being relatively prime to m, we have,

$\frac{c}{m} \approx \frac{1}{2}-\frac{1}{6}\sqrt{3}$, then we obtain a low value for the serial correlation, since the above value of $\frac{c}{m}$ is the root of the equation,

$$1-6\frac{c}{m}+6\left(\frac{c}{m}\right)^2=0$$

1.6 MULTIPLICATIVE CONGRUENTIAL GENERATOR.

Historically multiplicative congruential generators preceded mixed ones. Because their periods depend on X_0 and a -αs we shall see below - we want to select these quantities to guarantee a maximum period length.

The following theorem helps us to choose the parameters α, X_0 properly in order that the maximum period length be achieved.

Theorem F.

The maximum period possible when c=0 is 2^{e-2}, where 2^e is the computer size word.

This period is achieved if,

1) X_0 is an odd number;

2) α = 3 or 5 (modulo 8).

This theorem is due to R.D. Carmichael [Bull. Amer. Math. Soc. 16 (1910). 232-238].

So by taking the modulus $m=2^e$ the maximum period achieved is 2^{e-2} or more simply only $m/4=2^{e-2}$ numbers are available for use.

The choice of X_0 as an odd number and the multiplier α satisfying condition (2) guarantees that we run through the 2^{e-2} available numbers, but it is not clear how these numbers are distributed over $(0,2^e)$.

Because of this the full period mixed congruential generator appears more attractive.

In this report we will deal only with mixed congruential generators.

1.7 A BAD MULTIPLICATIVE CONGRUENTIAL GENERATOR.

Unfortunately a "low" correlation coefficient of a multiplier does not guarantee the randomness of the linear generator. Naturally a very good multiplier always has "low" correlation coefficient, but the converse is not always true.

Consider the multiplicative congruential generator

$$Z_{i+1}=\alpha Z_i \bmod m$$

where $\alpha = 2^{18}+3$, $m=2^{35}$.

It satisfies the maximum period criterion (See Theorem F) and comes close to minimize the correlation (see formula 4 page 19), but still remains a very bad multiplier!

We shall prove that the above multiplier is really bad. We may write,

$$Z_{i+1}=(2^{18}+3)Z_i \mod 2^{35}$$

$$Z_{i+2}=(2^{18}+3)Z_{i+1} \mod 2^{35} =$$

$$= (2^{18}+3)^2 Z_i \mod 2^{35}$$

$$= (2^{36}+3.2^{19}+9)Z_i \mod 2^{35} =$$

$$= (6(2^{18}+3)-2.9+9)Z_i \mod 2^{35}$$ (the term 2^{36} has vanished because it is a multiple of 2^{35}), and eventually,

$$Z_{i+2}=(6(2^{18}+3)Z_i-9Z_i) \mod 2^{35} =$$

$$= (6Z_{i+1}-9Z_i) \mod 2^{35}.$$

Recalling that $v_i=Z_i/2^{35}$ we also have the congruence relationship.

(1) $$v_{i+2}=6v_{i+1}-9v_i \pmod 1.$$

Suppose now that $0<v_i\leq\frac{9}{9}=0.1$.

Suppose again that,

(2) $$0<v_i\leq\frac{6v_{i+1}-\lfloor 6v_{i+1}\rfloor}{9}$$

From (2) we have $9v_i\leq 6v_{i+1}-\lfloor 6v_{i+1}\rfloor$ and,

(3) $6v_{i+1}-9v_i\geq\lfloor 6v_{i+1}\rfloor$ so $\lfloor 6v_{i+1}-9v_i\rfloor=\lfloor 6v_{i+1}\rfloor$

(4) If now $\frac{6v_{i+1}-\lfloor 6v_{i+1}\rfloor}{9}<v_i\leq 0.1$ we shall have

(5) $\lfloor 6v_{i+1}\rfloor > 6v_{i+1}-9v_i$ so $\lfloor 6v_{i+1}-9v_i\rfloor=\lfloor 6v_{i+1}\rfloor-1$.

Since $v_{i+1}=6v_{i+1}-9v_i-\lfloor 6v_{i+1}-9v_i \rfloor$ from (1) it is clear that for $0<v_i \leq \frac{6v_{i+1}-\lfloor 6v_{i+1} \rfloor}{9}$ we have,

(6) $0<v_{i+2}=6v_{i+1}-9v_i-\lfloor 6v_{i+1} \rfloor < 6v_{i+1}-\lfloor 6v_{i+1} \rfloor$, if we take into account equation (3).

Now for $(6v_{i+1}-\lfloor 6v_{i+1} \rfloor)/9<v_i \leq 0.1$ and taking into account relation (5), we shall have,

(7) $1>v_{i+2}=6v_{i+1}-9v_i-\lfloor 6v_{i+1} \rfloor+1 \geq 6v_{i+1}-\lfloor 6v_{i+1} \rfloor+0.1$

From (6) and (7) we have finally,

$v_{i+2}<6v_{i+1}-\lfloor 6v_{i+1} \rfloor$ or $v_{i+2} \geq 6v_{i+1}-\lfloor 6v_{i+1} \rfloor+0.1$.

The result shows that, for $0 \leq v_i \leq 0.1$, v_{i+2} is excluded from a sub-interval on the unit interval determined by the line $6v_{i+1} \bmod 1$ and $6v_{i+1} \bmod 1+0.1$.

So the sequence which is generated using the multiplier $2^{18}+3$ cannot be random.

Conclusions.

We have already discussed in some detail the method of generating random numbers in the range $[0,2^{35}-1]$ and the appropriate selection of the parameters α and c of the linear congruential generator.

The mixed procedure for generating random numbers on a binary machine with 36-bit word, (the first bit denoting the sign of the number) may be summarized as follows:

1. Choose as modulus m the number 2^{35}, since this makes the computation of $(\alpha X+c) \bmod 2^{35}$ fast and quite efficient.
2. Choose as initial value X_0 of the procedure any positive integer in the range $[0,2^{35})$.

3. The multiplier α should satisfy the condition $\alpha=5 \bmod 8$ in order to have the highest possible potency and maximum period.
   Also the multiplier α should be larger than $\sqrt{m}$ and smaller than $m-\sqrt{m}$. (See page 19).
   If the generator is to be used for high resolution Monte-Carlo problems, the multiplier must pass the spectral test. (See page 10).

   A table of some good multipliers which have passed this test is given on page A3.

4. The increment c of the generator must be an odd number.
   According to Greenberger's formula (page 19) the value of c which satisfies the condition $\frac{c}{m} = \frac{1}{2} - \frac{1}{6}\sqrt{3}$ will minimize the first order serial correlation between the pseudo-random numbers.

   This characteristic - that is the minimization of the serial correlation - is a highly desirable property of the random number generator as explained in page 18. But we must point out the deficiency of either the mixed or multiplicative generator in high resolution Monte-Carlo problems [12].

   MacLaren and Marsaglia were among the first who observed the weakness of the LCG in some tests - as maximum of t tests, which will be discussed in chapter 2 - and they proposed a new generator which is a combination of two linear congruential generators.

   The discussion of this generator will take place in the next chapter.

CHAPTER 2

COMBINATIONS OF CONGRUENTIAL GENERATORS

2.1 MARSAGLIA'S - MACLAREN METHOD.

Marsaglia and MacLaren were among the first to notice the bad statistical properties of the multiplicative generator with improper multipliers [12]. They suggested a new method which is a combination of two linear congruential generators.

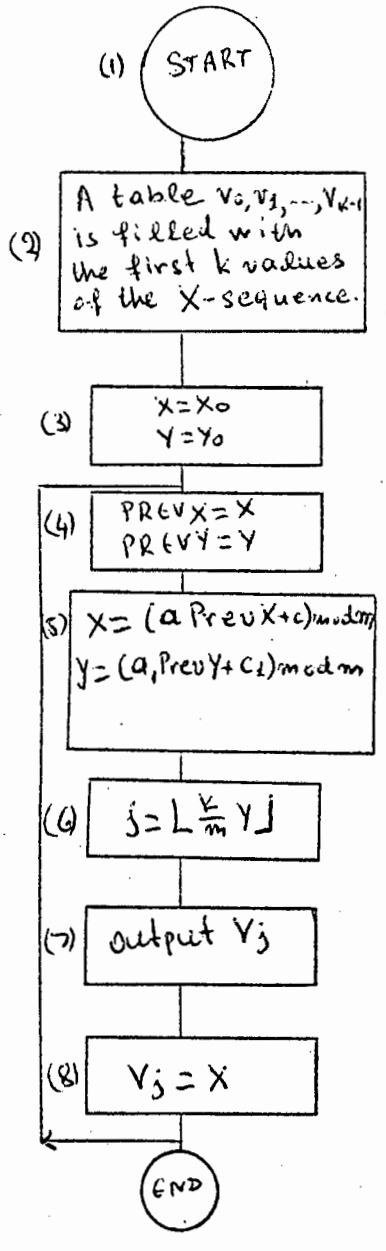
The values from the first generator are re-ordered randomly so as to form the output sequence. The second controls the reordering of the sequence.

Let us consider the generators,

(1) $X_{n+1} = (\alpha X_n + c) \bmod m$

(2) $Y_{n+1} = (\alpha_1 Y_n + c_1) \bmod m.$

An auxiliary table of K locations (K is usually taken to be (64 or 128) is filled with K values of the X-sequence. The Y-sequence determines which entry in the table is to be output as the next random number. The following flow-chart describes the technique.



In box 3 initial entry with actual parameters X_o, Y_o is provided. In Box 4, two temporary storage locations store the last used values of the two generators. In Box 5, two new random numbers are generated. In Box 6 a random integer j where $o \leq j < k$ is determined. This integer determines the table entry V_o to be output in Box 7. In Box 8 a new table entry replaces the entry just output.

In their original paper[12] Marsaglia and MacLaren used the generators

(3) $X_{n+1} = (2^{17}+3)X_n \bmod 2^{35}$

(4) $Y_{n+1} = (2^7+1)Y_n+1 \bmod 2^{35}$.

The value of k was taken to be 128. They compared their method with each of the above generators and they found it superior in all statistical tests they used to make the comparisons.

We observe that this is not surprising for generator (4) where they used a very bad multiplier indeed! This multiplier is too small to be efficient.

We have seen (page 19) that a multiplier should be larger than $\sqrt{m}$ and smaller than $m-\sqrt{m}$ where $m=2^{35}$.

From Greenberger's formula $C=\frac{1}{a}(1-6\frac{c}{m}+6(\frac{c}{m})^2)$ (page 19) if we replace c by 1 and α by 2^7+1 we find the serial correlation coefficient

to be equal $\frac{1}{2^7+1} \simeq \frac{1}{129}$. This value is too high to be good.

Since the multiplier α does not satisfy the condition $\alpha \equiv 5 \bmod 8$, neither the maximum potency is achieved (it is only 5) nor the maximum period length is attained.

This is the reason why they found rejectable values when they applied the frequency, serial, maximum of 2 etc. tests to this generator.

But the poor performance of the second generator proves the power of their method!

When the above generators were combined and tested with a variety of statistical tests, they found out [12] that the statistical performance of their method was superior compared with each generator separately.

The main drawback of this method is the time required to generate a sequence of random numbers. It is easily seen that this is twice as long as it takes to generate the sequence (1) alone.

We try in this report to alleviate this problem by modifying slightly the original Marsaglia - MacLaren method.

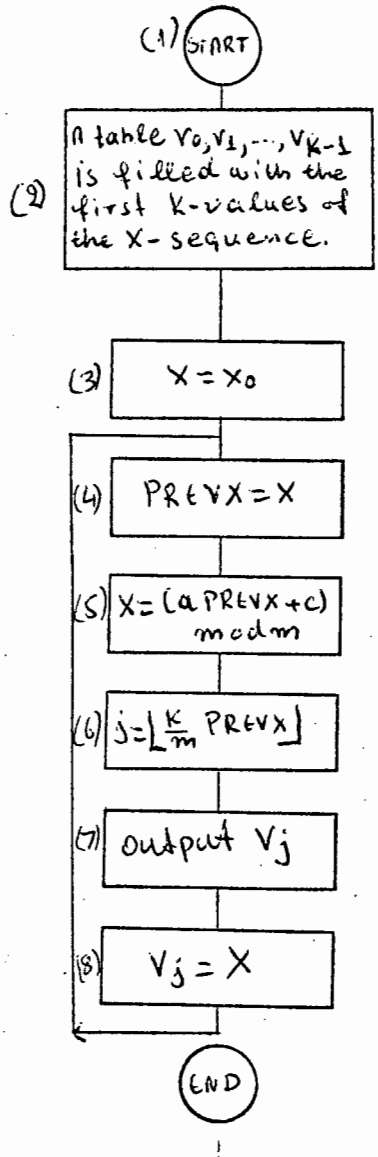
We shall show that this modified generator is preferable to the original one for two reasons.

1. the modified generator is faster than the original one, and
2. the new generator improves the randomness of the original generator.

2.2 A MODIFIED GENERATOR.

In order to reduce the CPU time required to generate a random sequence using Marsaglia - MacLaren's method, we do not use a second generator, but use the X-sequence to control the output position as well as the value.

The output position j is determined by the value of the previous term in the generator {Xm}. The following flow-chart describes the technique.



In Box 3 a starting value for the X-generator is provided.

In Box 4 a temporary storage location stores the last used value of the X-generator.

In Box 5 a new random number is generated.

In Box 6 a random integer j where $0 \leq j < k$ is produced determined by PREVX. This integer determines the table entry V_j to be output in Box 7.

In Box 8 a new table entry replaces the entry just output.

In this report the value of K is taken to be 128 and the modulus m equal to 2^{35}.

A subprogram which puts in action the above procedure is described on the next page.

IRAND - Modified Generator.

Purpose.

This function sub-program generates a random sequence of integers from the interval $[0,2^{35})$ using the modified Marsaglia - MacLaren method.

Entry.

IRAND has three entry points: two are for normal usage; one is for user modification of the basic IRAND routine.

Initialization entry.

CALL PINAX(MS)

where:

MS is an arbitary starting number (integer) for a table of 128 pseudo-random numbers.
On return to the calling program MS contains the last number in this table. (INPUT, OUTPUT).

Standard entry.

I=IRAND(MS)

where:

IRAND is the name of the sub-program and contains the result of the random number generator upon exit from the sub-program.

MS is an arbitrary starting number. On succeding calls, MS contains the last number generated by the auxiliary generator

In this report, we always take as starting number of the sub-program IRAND, the last number of the table, which is given by the entry point PINAX. (INPUT, OUTPUT).

Entry for sub-program modification.

CALL SETP(M,N)

where:

M is the multiplier for the auxiliary generator . (INPUT).

N is the increment for the auxiliary generator . (INPUT).

Programming method.

This sub-program was coded in Univac's 1108 Assembler language.

Restriction.

None.

Error returns.

None.

Other sub-programs required.

None.

Special considerations.

In order to generate a sequence of random integers, the following logical sequence of instructions may be used:

```
      DIMENSION IR(n)
       MS = i
      CALL PINAX(MS)
       DO 1 L = 1,n
    1    IR(L) = IRAND(MS)
```

SETP as we have seen is not a separate routine; it is an entry point within IRAND. The user can ignore the SETP entry if he wishes. The purpose of this entry is to enable IRAND to become any linear congruential generator by modification of IRAND's multiplier and increment.

The multiplier, and increment c, of the auxiliary generator can be set (via SETP) to any integers between 0 and 2^{35}. If SETP is not called, then the values that will be used are α=271828121 and c=7261067265. The PINAX entry must be used to initialize the table from which all the other random numbers are generated.

Comments.

For a coding of the sub-program IRAND see page C1.

2.3 COMPARISON OF THE TWO METHODS.

The IRAND sub-program represents the modified Marsaglia-MacLaren method. Univac's 1108 MRAND generator represents the original MacLaren method.

We describe briefly MRAND generator.

MRAND has three entry points:

a) Entry for sub-program modification.

CALL SETM(α1,c1,α2,c2)

where α1 = multiplier of the first generator, c1 = increment for the first generator, α2 = multiplier for the second generator and c2 = increment for the second generator.

b) Initialization entry.

CALL TABLE(MS)

This entry is analogous to the entry point PINAX (see page 29)

c) Standard entry.

I=MRAND(M,N).

where M,N are the starting numbers for the auxiliary generators. On succeding calls, M and N contain the last numbers generated by the auxiliaries generators of MRAND.

In this report we will try to compare these two generators in respect of the speed of generation of random sequences of equal lengths and in respect of their statistical performance.

The latter is done in chapter 3. IRAND's multiplier α is taken to be equal to 2718281821 (see table 3 page A3). This multiplier passes the spectral test and the values of C2,C3,C4 are 2.59, 1.15, 1.75 respectively.

As an increment c we take the number 7261067265 which satisfies Greenberger's formula (see page 19).

MRAND's multipliers and increments are taken as follows:

1st generator $\alpha 1$= the same as IRAND's multiplier.

c1= the same as IRAND's increment.

2nd generator $\alpha 2$= 3141592653

c2= C1

The multiplier $\alpha 2$ also passes the spectral test (see table 1, page A3).

As far as the speed of generation is concerned thirty thousand random numbers were generated by both generators.

The CPU time required for the generator IRAND to produce them was 1713 miliseconds while for the MRAND was 2637 miliseconds (for program and output results see pages C2,C3). We see that a 35 percent, reduction of CPU time was achieved.

Chapter 3 now examines the statistical performance of both generators.

CHAPTER 3

3.1 STATISTICAL TESTS OF RANDOM NUMBERS

The statistical properties of random numbers generated by the method outlined in the previous chapter should coincide with the statistical properties of numbers generated by an idealized chance device that selects numbers from the interval (a,b) where $0 \leq a,b \leq 2^{35}-1$, independently and with all numbers equally likely. Clearly, the pseudo-random numbers produced by computer programs are not random in this sense, since they are completely determined by the starting data and have limited precision. But so long as our pseudo-random numbers can pass the set of the statistical tests implied by the aforementioned idealized chance device, these pseudo-random numbers can be treated as "truly" random numbers even though they are not.

The following statistical tests are among the more important tests for randomness cited in the literature [9].

3.2 CHI-SQUARE GOODNESS OF FIT TEST

The Chi-square test is perhaps the best-known of all statistical tests. Let us define the statistic,

$$(1)\quad X^2 = \sum_{1 \leq s \leq K} \frac{(y_s - np_s)^2}{np_s} \quad , \text{ where}$$

K = number of categories, y_s = the number of observations which fall into category s, p_s = the probability that each observation falls into category s and n = the total number of independent observations, then X^2 is called the chi-square statistic. The number of degrees of freedom is K-1, one less than the number of categories. A table of the chi-square distribution is given on page A1. This table gives values of the X^2 with ν degrees of freedom. For example if we

look in row with the $\nu = 1$ we can see that 0.00016 corresponds to 99 per-cent entry. The meaning of that is as follows:
"The quantity X^2 will be greater than 0.00016, 99 per-cent of the time."
This indicates that the observed values are too close to the expected values. If we look at the values 6.635 we see that the quantity X^2 will be greater than this value only one per-cent of the time. This represents a significant departure from the expected values.
We can describe the chi-square test as follows:
A fairly large number, n, of independent observations is made. We count the number of observations falling into each of K categories, and compute the quantity X^2 given in equation (1). Then X^2 is compared with the numbers in Table I(page A1) with $\nu = K - 1$. If X^2 is less than 99% or greater than the one-per-cent entry, we reject the numbers as not sufficiently random. For example if X^2 is less than 99 per-cent entry this indicates that the observed values are so close to the expected values, we cannot consider the result to be random!

If X^2 lies between 99 and 95 per-cent or 5 and 1 per-cent, the numbers are "suspect". If X^2 lies between 95 and 90 per-cent, or 10 and 5 per-cent, the numbers might be 'almost suspect'.

If we apply the chi-square test at least three times on different sets of numbers, and if at least two of the three results are suspect the numbers are considered as not sufficiently random and therefore are rejected. We apply these ideas to a variety of statistical tests, which will judge the randomness of both IRAND and MRAND generators.

3.3 EQUIDISTRIBUTION OR FREQUENCY TEST

One of the more usual tests which is applied to random integer sequences in order to investigate whether the sequence is uniformly distributed between 1 and d is the frequency test.

In general each test is applied to a seuqence V_0, V_1, ... of real numbers, which purports to be uniformly distributed between zero and one. This test is designed for integer-valued sequences, instead of real-valued sequences. We shall examine if integer valued-sequences between 1 and d-where d is a power of 2 in a binary machine, are uniformly distributed in this range. It is a simple matter to convert a fractional sequence V_1, V_2, ... to an integer valued sequence, where we wish the integer values of the sequence to be located between 1 and d. We simply form the sequence V_1, V_2, ... where $V_n = \lfloor dV_n \rfloor + 1$.

To make this test we count the number of times $V_i = r$ for $1 \leq i \leq n$ where $1 \leq r \leq d$ and then we apply the chi-square test, using K (number of categories) = d, and probability $p = \frac{1}{d}$ for each category, since if the above sequence was really uniformly distributed between 1 and d the theoretical probability would be 1/d. The degrees of freedom are $\nu = K-1 = d-1$. The computer program which puts into action the above procedure was given the name TEST 1.

PROGRAM TEST 1

Programming Method

TEST 1 is coded in Fortran V.

Nomenclature

NTEST	is the number of tests performed.
NN	is the number of random numbers tested.
NG	is the number of groups
NS	is the starting number of IRAND generator
NF	is the number of degrees of freedom
V	is the chi-square statistic
VAR	is the percentage of the time the qui-square statistic is greater than value V.

Other Subprograms Required

IRAND function

CHI function

Supporting Information

CHI is a library routine[19] which computes the probability of getting a chi-square value less than V. The input to this function is the variable V and the number of degrees of freedom NF.

Comments

The following results were obtained using the default multiplier and increment of IRAND (see page 31). TEST I was executed ten times to test IRAND for uniformity. In each of the five tests, 100 numbers were sorted in 5 groups. In each of the last five 1000 numbers were sorted into 50 groups. Details of the output of TEST I can be found in our appendix on page C5 . The only "almost suspect" value is the value 5.63%. The other nine tests were passed successfully. For TEST I program see page C4 .

TEST1 Input

TEST 1 input data has the following format.

Card 1	FORMAT(I5)
col 1-5	NTEST, the number of tests to be performed.
Card 2	FORMAT (4X,I4,I12,I2)
col 5-8	NN is the number of pseudo-random numbers to be generated by IRAND.
col 9-20	NS is the starting number of the generator.
col 21-22	NG denotes the number of groups.

TEST1 INPUT DATA

```
10
 100              1111          5
 100              1112          5
 100              1113          5
 100         2468134327         5
 100         2468135790         5
1000               928          50
1000               929          50
1000               930          50
1000               931          50
1000         2468134325         50
```

The same program TEST 1 was run using in the place of IRAND generator, Univac's 1108 random number generator MRAND. (For output results and program see pages C7,C6.). The entry SETM(M1,N1,M2,N2) of the f function MRAND is called for parameters modification.

The multipliers M1 and M2 and increments N1 and N2 were set equal to their corresponding values of the page ...32 . We observe from the output results that One test gave a "reject" value -

99.13% - and another one as "almost suspect" value, namely 8.23%

Concluding this discussion we can say that both generators passed the frequency test successfully.

MRAND TEST1 INPUT DATA

10

100	1111	4536	5
100	1112	5849	5
100	1113	67930	5
100	2468134327	1765437890	5
100	2468135790	2468135180	5
1000	928	180	50
1000	929	456	50
1000	930	982467	50
1000	931	831	50
1000	2468134325	7594325678	50

3.4 WEAKNESS OF THE EQUIDISTRIBUTION (FREQUENCY) TEST

This test does not intend to be a strong criterion for randomness of our generator for the following reasons:

Let u and v be real numbers, $0 \leq u < v \leq 1$. If V is a random variable which is uniformly distributed between zero and one, the probability that $u \leq V \leq v$ is $P(u \leq V \leq v) = \int_u^v dx = v - u$.

For example if $u = 0$, $v = \frac{1}{2}$ the probability that V_i, is between $0, \frac{1}{2}$ is $v - u = \frac{1}{2}$.

Now as infinite seuqence $V_0, V_1, \ldots$ is said to be equi-distributed if the following relationship holds:

$$\lim_{n \to \infty} I(n)/n = u - v,$$

where $I(n)$ is the number of values of j, $0 \leq j < n$ such that $u \leq V_j < v$ for all real numbers u,v $0 \leq u < v \leq 1$. The ratio $I(n)/n$ is apparently the probability $P(u \leq V_n \leq v)$. Now we will give an example of a sequence which is equidistributed without being random. Let the sequences $V_0, V_1, \ldots$, $U_1, U_2, \ldots$ which we suppose are equidistributed, and let $w_0, w_1, \ldots = \frac{1}{2} V_0, \frac{1}{2} + \frac{1}{2} U_1, \frac{1}{2} V_1, \frac{1}{2} + \frac{1}{2} U_1, \ldots$. This sequence is also equidistributed, since the sequence $\frac{1}{2} V_0, \frac{1}{2} V_1, \ldots$ is equidistributed (see proof below) between 0 and $\frac{1}{2}$ while the alternative terms are equidistributed between $\frac{1}{2}$ and 1.

In the sequence of w's a value less than $\frac{1}{2}$ is always followed by a value greater than or equal to $\frac{1}{2}$ and conversely. The sequence w has a very simple pattern to be considered as a random sequence.

Proof why the sequence $\frac{1}{2} V_1, \ldots$ is equidistributed between $0, \frac{1}{2}$.

Since $V_0, V_1, \ldots$ is equidistributed between 0,1 then

(1) $P(u \leq V_n \leq v) = v - u$ for all u,v between 0,1.

From (1) we also have $P_1(\frac{u}{2} \leq \frac{1}{2} V_n \leq \frac{v}{2}) = \frac{v}{2} - \frac{u}{2}$ and if we put $u = 0$ and $v = 1$ we have the relation

$P_1(0 \leq \frac{1}{2} V_n \leq \frac{1}{2}) = \frac{1}{2}$.

3.5 SERIAL TEST

Program TEST2, tests pairs of integers (X_{2j}, X_{2j+1}) of numbers generated by IRAND to see if they behave as though drawn independently from a population of uniformly distributed pairs of integers.

To make this test we count the number of times the pair $(X_{2j}, X_{2j+1}) = (q,r)$ occurs for $1 \leq j < n$, $X_n = \lfloor dU_n \rfloor + 1$ where $0 \leq U_n < 1$.

These counts are to be made for each pair of integers (q,r) with $1 \leq q,r < d$, that is the region of the plane into which the pairs can fall is divided into squares of equal area and the number of pairs falling in each square is counted. The number of categories are now $K = d^2$ and each pair has probability $1/d^2$ to fall into a certain square. So we apply the chi-square test to the above categories with probability $1/d^2$.

Contrary to the frequency test, we chose d to be a smaller number in order to avoid having too many categories. A valid chi-square test should have n (number of observations) greater than 5K where K is the number of categories [11].

PROGRAM TEST2

Programming Method

TEST2 is coded in Fortran V.

Nomenclature

NT	is the number of tests performed.
NP	is the number of pairs, (X_{2j}, X_{2j+1}).
NG	is the number of groups.
MS	is the starting value of the generator IRAND.
NF	is the number of degrees of freedom.
MA(I,J)	is an array which is used as a counter. It contains the number of points which fall in a certain square.
V	is the qui-square statistic.
VAR	is the percentage of time the qui-square statistic is less than value V.
NGSQ	is the number of categories.

Other Subprograms Required

IRAND function

CHI function

Supporting Information

As in frequency test.

Comments

TEST2 was executed 12 times to test for uniformity of pairs of numbers. In each of the first four tests 60 pairs of numbers were sorted in 4 groups; in each of the next four 200 pairs were sorted into 16 groups; in the last four, 2000 pairs were sorted into 100 groups. From the output (see page C9) we can see that only two tests gave "almost suspect" results, namely 93.8% and 5.98%. The other 10 tests were passed successfully. For the TEST2 program see page C8.

TEST2 INPUT

Card 1	FORMAT (I5)
col 1-5	NT, the number of tests to be performed.
Card 2	FORMAT (2I5, I12)
col 1-5	NP, the number of pairs of random numbers to be generated by IRAND.
col 6-10	NG, is the number of groups.
col 11-22	NS, the starting number of generator IRAND.

TEST2 INPUT DATA

See next page

12		
60	2	19
60	2	23
60	2	12345
60	2	543212
200	4	5
200	4	9
200	4	3141592
2000	10	122070314
2000	10	117
2000	10	131
2000	10	8924681
2000	10	1591917152

The same program was run using MRAND in the place of IRAND generator. From the output of MRAND (see page C10) we see that one test gave the 'reject' value 0.629; while the other 11 were passed successfully. The same data have been used as previously. For the second generator of MRAND the following starting values were used:

MS1
81
123
3789
643212
13
1
3200000
222445682
114
132
9990035
2490316794

3.6 RUN TEST

A sequence of numbers may be tested for "run up" and "run down". This means we examine the length of monotone subsequences of the original sequence i.e. subsequences which are increasing or decreasing.

Let us consider the sequence of ten numbers,

1,2,9,8,5,3,6,7,0,4. Putting a vertical line at the left and right between X_j and X_{j+1} whenever $X_j > X_{j+1}$ we obtain

|1,2,9|8|5|3,6,7|0,4| which displays the runs up. Putting now a vertical line of the original sequence at the left and right and between X_j and X_{j+1} whenever $X_j < X_{j+1}$ we obtain |1||2| 9853|6|70|4| which displays the runs down.

We observe that there is a run of length 1 followed by a run of length 1, followed by a run of length 4 followed by a run of length 1 followed by a run of length 2 followed by a run of length 1.

Let us consider the sequence,

NX(1), NX(2), ..., NX(LG) (1) of length LG and let us put

MIN = $\lceil \frac{LG+1}{2} \rceil$. Then the probability, P, that the sequence contains a run up or down of length greater or equal to LG/2 is given by the formula [11].

(2) P = 2*[(MIN+1)*LG -(MIN**2+MIN-1)]|(MIN+2).

This is the probability of success to have a run up or down of length greater or equal to LG/2, if the sequence (1) consisted of numbers really random. The run test is made by taking a variety of samples from the examined sequence of random numbers, each sample containing a certain number of subsequences of length LG. We count the total number of runs (up and down) of length greater than or equal to LG/2 and we apply the function BIN (see below) which calculates the probability that a random variable binomially distributed is less than or equal to a given value.

(Here the "given value" is the actual number of runs up and down).

PROGRAM TEST4

Purpose

TEST4 tests the distribution of runs of numbers. TEST4 calls IRAND to generate a sequence,

$(X1, \ldots, X_{LG}), (X_{LG+1}, \ldots, X_{2LG}), \ldots, (X_{(NG-1)(LG+1)}, \ldots, X_{(NG)(LG)})$

where LG (length of the subsequence) and NG (number of groups) are set by the user. TEST4 then calculates the total number of runs of length greater than LG/2 that occur among the NG different groups. Using the binomial probability model, the result is compared with the result that is expected under the assumption that the sequence is random.

Programming Method

TEST4 is coded in Fortran V.

Nomenclature

N	is the total number of tests
NG	is the total number of groups observed.
LG	length of each group.
NS	starting number of IRAND.
MIN	equals to $\lceil \frac{LG}{2} \rceil$
NX(1),..., NX(LG)	is a group of LG consecutive numbers generated by IRAND.
N COUNT	contains the total number of runs of length greater or equal to LG/2 at the end of the computation.
NU	is used to store the length of the runs up.
ND	is used to store the lengths of the runs down.
V	denotes the expected number of runs if the numbers were truly random.

P denotes the probability of success to have a run up or down of length greater than or equal to LGI2.

PL is the probability that a random variable binomially distributed with known parameters is less than or equal to a given value.

Other Subprograms Required

1) IRAND function.

2) BIN function. This is a library function (6) which calculates the probability PL.

The calling sequence of this function is,

PL = BIN (M,N,P)

where,

BIN is the name of the function subprogram containing the computed value of the binomial probability on return to the calling program.

M is the variable whose cumulative probability is completed.

N is the number of observations in the sample.

P is the probability of success.

Comments

TEST4 is executed 39 times to test the occurrence of runs in IRAND. In each of the first thirteen tests, 500 sequences of length 3 were tested for runs of length 2; in each of the next thirteen tests, 1000 sequences of length 5 were tested for runs of length 3 or greater; in each of the last thirteen tests 3000 sequences of length 10 were tested for run of length 5 or greater. The results are presented in page C12. Three of them are 'almost suspect', namely 6.2551%, 8.1945, 94.4%, three 'suspect', namely 3.4508%, 98.1292%, 1.9851% and one 'reject' namely 99.5104%.

The other 32 tests were passed successfully. Generally the behaviour of IRAND with regard to this test is satisfactory.

For a Flow-chart and program see pages BI, CII.

TEST INPUT

TEST4 input data has the following format.

card 1 FORMAT(I5)

col 1-5 N, the number of tests to be performed.

card 2 FORMAT (2I5, I12)

col 1-5 NG, the number of groups of numbers to be generated.

col 6-10 LG, the length of each group.

col 11-22 MS, starting number of the generator IRAND.

TEST4 INPUT DATA (see page CI4).

The same program TEST4 was run using instead of IRAND the function MRAND. The results of this test are presented in page CI3. We see from these results that 5 of them are 'almost suspect' namely 7.5098%, 90.5298%, 90.0454%, 5.9648%, 5.9648% and one 'suspect' namely 96.0363% while the rest 33 tests were passed successfully.

MRAND TEST4 INPUT

Same format as IRAND's.

MRAND TEST4 INPUT DATA

The same as previously. As a seed for the secnd auxiliary generator of MRAND the last number generated from subroutine TABLE, this is an entry point to MRAND and corresponds to the entry point PINAX of IRAND, was used.

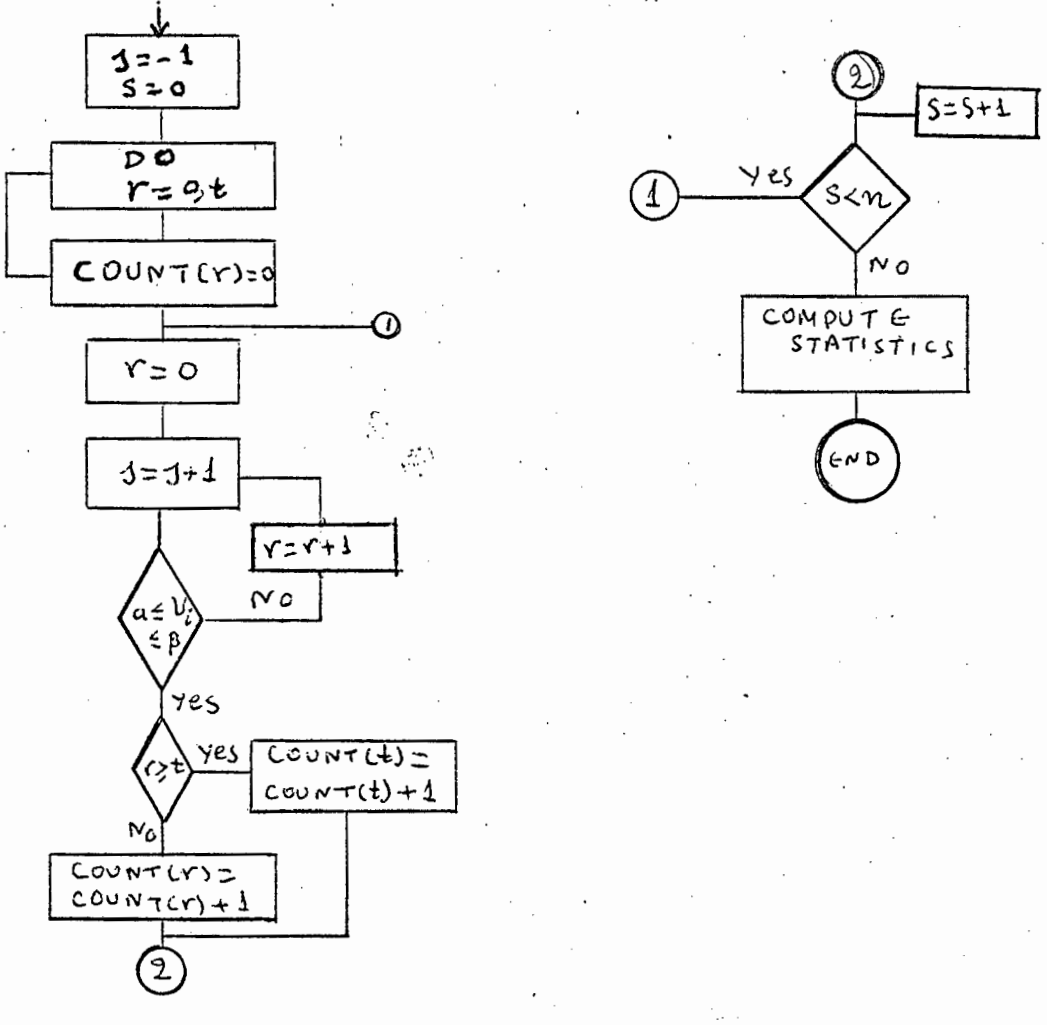
3.7 GAP TEST

Let us consider the sequence;

$V_0, V_1, V_2, \ldots$.(I) of real numbers, which purports to be uniformly distributed between zero and one. Let us assume two real numbers a,b where $0 \leq a < b \leq 1$ and a subsequence of the original sequence $V_j, V_{j+1}, \ldots, V_{j+r}$ in which V_{j+r} lies between a and b but the other V's do not. We say that a 'gap' of length r has been found. For example if we have the sequence V_0, V_1, V_2, V_3 where $a \leq V_3 \leq b$ and the numbers V_0, V_1, V_2 do not lie in the range (a,b) then the gap of the above sequence of numbers has length 3. An efficient algorithm in order to tabulate a gap of length r is given below.

Algorithm A.

The following algorithm applied to the sequence (I), counts the number of gaps of length $0,1,2,\ldots, t-1$ and the number of gaps of length $\geq t$, until n gaps have been tabulated.



From the above algorithm it is clear that COUNT(r) contains the gaps of length r, while COUNT(t) contains the gaps of length t.

After this algorithm has been performed and n gaps have been tabulated, the chi-square test is applied to the K = t + 1 categories, COUNT (0),...., COUNT(t) using the following probabilities:

$P_0 = P$, $P_1 = P(1-P)$, $P_2 = P(1-P)^2$,...., $P_{t-1} = P(1-P)^{(t-1)}$

$Pt = (1-P)^t$, where p = b-a. Here p = b-a is the probability that $a \leq V_j < b$.

The above probabilities can be easily derived if we note that a gap of length r means, than r numbers are outside the range (a,b) while the (r+1)th number lies in the range (a,b). So the event to have a gap of length r, consists from two simple events.

1. r numbers to be outside the interval (a,b) → probability $(1-P)^r$ and
2. the (r+1)th number to be inside the interval (a,b) → probability P.

So the probability a gap of length r occurs is $P_r = P(1-P)^r$.

Program TEST5

Purpose

TEST 5 examines the length of gaps between occurrences of V_j in a certain range (a,b).

Nomenclature

NN	is the total number of tests.
A	is the left point of the interval (a,b).
B	is the right point of the interval (a,b).
T	is the maximum length of a gap.
N	is the number of gaps desired.
COUNT	is an array which holds the total number of gaps of length 0,1,...., t-1, t.

D is the number of categories.

NF is the number of degrees of freedom.

PI is the probability to have a gap of length 0. PI equals to B-A.

P is an array. The elements of the array denote the probability $P_2, P_3, \ldots\ldots, P_t$.

VV is the chi-square statistic.

VAR is the probability of getting a chi-square value greater than V.

Other Subprograms Required

IRAND function.

CHI function.

Comments

Twenty different sequences of random numbers were examined for gaps of various length and for various intervals. The output IRAND is given in page C16 . From the output results we observe that four are 'almost suspect' namely 7.66%, 8.929%, 91.696%, 92.042% and four 'suspect' namely 96.626%, 96.57%, 97.877%, 97.639 while twelve tests were passed successfully. Concluding we can say that the "gap" test is passed by IRAND although the found chi-square values are not quite good..

For a coding of TEST5 see page C15.

TEST INPUT

TEST5 input data has the following format.

Card 1 FORMAT (I5) .

col 1-5 NN is the total number of tests.

card 2 FORMAT (2F3.2, I2, I4, I12)

col 1-3 A is the left point of the interval (A,B)

col 4-6 B, is the right point of the interval.

col 7-8 T is the maximum length of a gap.

col 9-12 N, the number of gaps desired.

col 13-24 MS is the initial number of generator IRAND.

TEST5 INPUT DATA

See page C17.

The same program was run using instead of IRAND the MRAND generator. The output results are presented in page C18. From the output results we observe that two of them are 'almost suspect' namely 5.318%, 92.735% and one 'reject' namely 0.333% while the remaining 17 tests were passed successfully. We can see that MRAND has much better behaviour than IRAND in respect to this test. The input data are exactly the same as above. As initial number for the first generator of MRAND the last number generated by the entry point TABLE was used, while for the second generator the input value MS was used.

3.8 POKER (PARTITION) TEST

The Poker test considers n groups of five successive integers, $(V_{5j}, V_{5j+1}, \ldots\ldots, V_{5j+4})$ $0 \leq j < n$.

We observe which of the following seven patterns each quintuple matches:

All different:	abcde	Full house:	aaabb
One pair :	aabcd	Four of a kind:	aaaab
Two pairs :	aabbc	Five of a kind:	aaaaa
Three of a kind:	aaabc		

A chi-square test is based on the number of quintuples in each category. A simpler version of this test is simply to count the number of distinct values in a set of five. We would have five categories:

5 different = all different

4 different = one pair

3 different = two pairs or three of a kind.

2 different = full house, or four of a kind.

1 different = five of a kind.

In general we can consider n groups of K successive numbers, and we can count the number of K-tuples with r different values. A chi-square test is then made, using the probability,

(1) $$P_r = \frac{d(d-1)\ldots(d-r+1)}{d^k} \quad \left\{ {k \atop r} \right\}$$

that they are r different.

The numbers $\left\{ {k \atop r} \right\}$ are the Stirling numbers of the second kind (see page 57)

Since the probability P_r is very small when r = 1 or 2 we can lump a few categories together before the chi-square test is applied.

In order to derive the above formula we must prove first that $\left\{ {k \atop r} \right\}$ is the number of ways to partition a set of k elements into exactly r parts. For example the set {1,2,3,4} can be partitioned into two subsets in $\left\{ {4 \atop 2} \right\} = 7$ ways:

{1,2,3},{4};{2,4},{3};{1,3,4},{2};{2,3,4},{1};{1,2},{3,4};

{1,3},{2,4};{1,4},{2,3};

we shall use the fact that $\left\{ {k \atop r} \right\} = r\left\{ {k-1 \atop r} \right\} + \left\{ {k-1 \atop r-1} \right\}$ (2).

Let f(k,r) be the number of partitions of {1,2,.....,k} into r parts.

Clearly f(1,1) = 1.

If k > 1 the partitionings are of two varieties.

(a) The element k alone forms a set of the partition; there are f(k-1,r-1) ways to construct partitions like this.

(b) The element k appears together with another element; there are r ways to insert k into any r-partition of {1,2,...., k-1}, hence there are rf(k-1,r) ways to construct partitions like this. We therefore conclude, $f(k,r) = f(k-1,r-1) + rf(k-1,r)$ and by induction - taking into consideration formula (2) - we have $f(k,r) = \left\{ {k \atop r} \right\}$.

So in order to derive the formula for p_r, we must count how many of the d^k k-tuples of numbers between 0 and d-1, - where d is an integer $\geq k$ - have exactly r different elements, and we divide the total by d^k. Since the number of ordered choices of r things from a set of d objects is given by the formula $d(d-1) \ldots (d-r+1)$ (3) and the number of ways to partition a set of k elements into r parts by the formula $\left\{ {k \atop r} \right\}$ (4) then the number of ways the d^k k-tuples of numbers between 0 and d-1 to have exactly r different elements is found if we multiply the relations (3) and (4). So the formula (1) for p_r is fully justified

A flow-chart for this test is given on page 54.

In Box 2 the categories CATEG are initially set equal to zero.

Similarly in boxes 5 and 6 the variables OCCURS and DIF are set equal to zero.

In Box 8 we generate a random number U uniformly distributed between zero and one.

In Box 9 we find a random integer L, where $1 \leq L \leq d$.

In Box 10 we ask if OCCURS (L) is equal to zero. If yes we put OCCURS (L) equal to 1 (any number will work) and increase DIF by one. This means that we found a distinct element in the range [1,d]. If now the answer is no, this means that we found an element L, equal to the previous ones (not different) and we generate another random variable U (we go to box 8) and the process is repeated until the generated numbers U become equal to k.

Then we go to box 15 where we increase CATEG (DIF) by one.

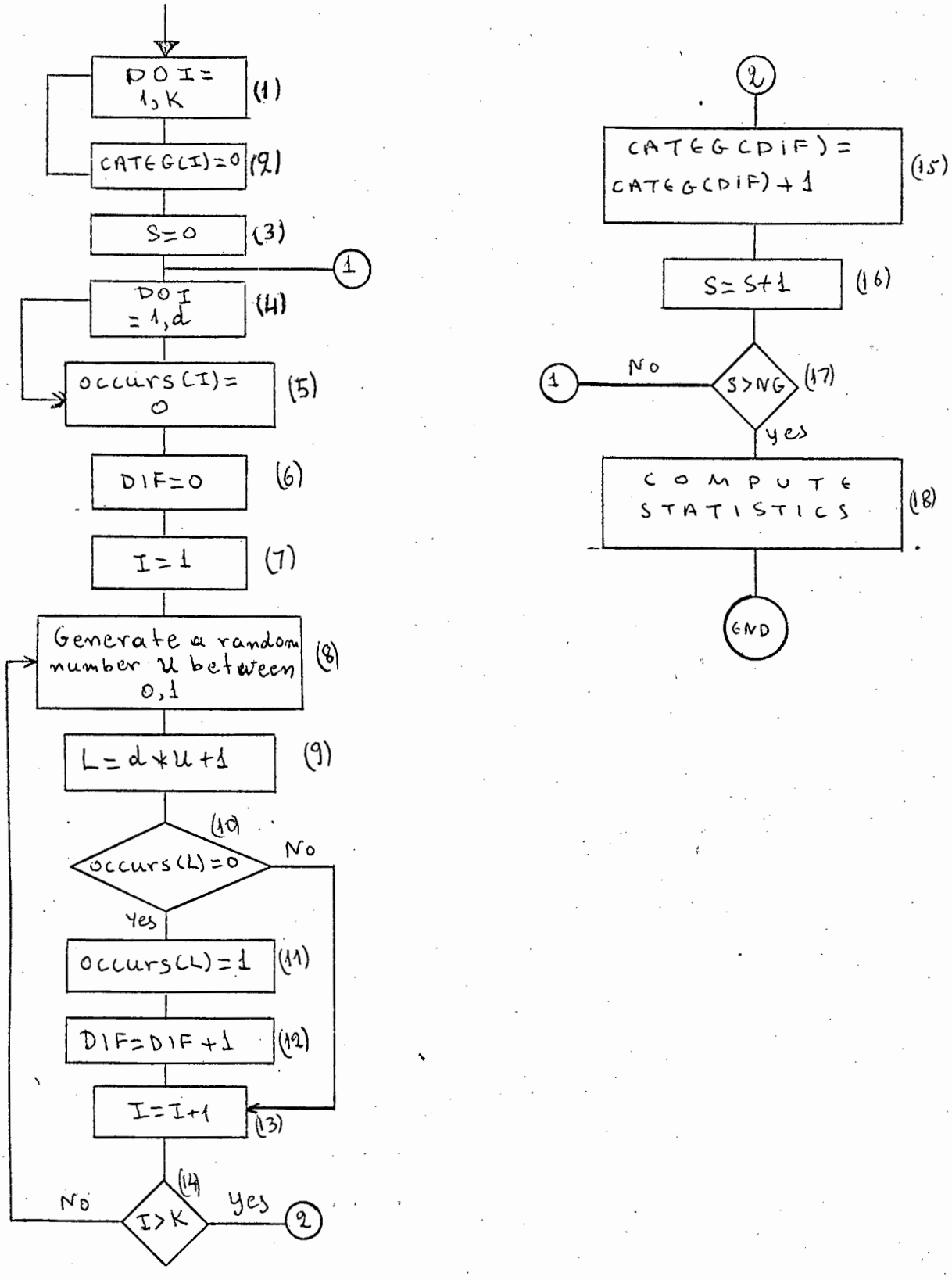


Figure: Algorithm of Poker test.

In box 17 we ask whether the group of random numbers under examination are exhausted and if no, the whole process is repeated again (we go to box 4) and if yes we compute the chi-square statistic.

NOTE: the length of each group is K.

The computer program which puts into action the above procedure was given the name TEST 8.

Programming Method

TEST 8 is coded in Fortran V.

Nomenclature

The variables which appear in the program have been previously defined.

Other Subprograms Required

1) Function KORDER to calculate the number of ordered choices of r things from a set of d objects. (see supporting information)
2) Function ISTIR. This function calculates the Stirling numbers of the second kind. (see supporting information)
3) IRAND function.
4) CHI function.

Comments

The output from IRAND is presented in page C20.

From the output we observe then two values are 'suspect' namely 98.77%, 95.02%; three values 'almost suspect', namely 9.62%, 5.43%, 94.51% and one 'reject' namely 0.37%. The other 10 tests were passed successfully. For a coding of TEST 8 see page C19.

TEST 8 INPUT

TEST 8 input data has the following format:

card 1	FORMAT(I5)
col 1-5	NN is the number of tests performed
card 2	FORMAT(I8, I4, I12, I2)

col 1-8 NG denotes the number of groups which will be tested.

col 9-12 D. The random numbers which are generated by IRAND are subsequently converted into integers, which lie in the range [1,D] .

col 13-24 MS is the seed for the generator IRAND.

col 25-26 T is the group length.

TEST INPUT DATA

16

300	6	1	5
300	6	12398	5
300	7	3452890657	5
300	7	1276054768	5
1400	8	444098906	6
1400	8	666666	6
1600	9	554987	6
2000	9	44388899	6
6000	10	553376	7
6000	10	438767	7
14000	11	321232	7
14000	11	200900000	7
60000	12	2	8
60000	12	33	8
90000	13	444	8
90000	13	288847	8

The same test was run using MRAND instead of IRAND.
The results of this test are presented in page c21.
Four of them are 'suspect' namely 2.00%, 96.31%, 97.09%, 96.88% and two 'almost suspect' namely 5.73%, 90.58%. The other 10 tests were passed successfully by MRAND. We can see from these results that MRAND behaves slightly better than IRAND in respect of this test. Format and input data are exactly the same as in the first test for IRAND.

As initial number for the first auxiliary generator for MRAND, the last number generated by the entry TABLE was used, while for the second generator the input value MS was used.

Supporting Information

As we have previously seen, it is necessary to find the number of ordered choices of L things taken from a set of N objects. Subprogram KORDER below does it for us.

Subprogram KORDER

Entry

Function KORDER has one entry:

M = KORDER (N,L)

where:	Description
N	is the number of elements in set of objects. Integer (INPUT)
L	is the number of things which are taken from the N objects. Integer (INPUT).

The output variable M equals to

$$N(N-1)(N-2) \ldots (N-L+1).$$

Restrictions

L must be less or equal to N.

Error Returns

None.

Other Subprograms Required

None.

For a coding of KORDER see page C29

FUNCTION ISTIR

Purpose

To compute the stirling numbers of the second kind. We denote these numbers by the symbol $\{{n \atop k}\}$. We give some important identities

involving Stirling numbers.

(1) $\sum_{k} \binom{n}{k} k^m(-1)^k = (-1)n! \left\{ {m \atop n} \right\}$ where $1 \leq k < n$.

(2) $\left\{ {n \atop m} \right\} = m \left\{ {n-1 \atop m} \right\} + \left\{ {n-1 \atop m-1} \right\}$ if $n > 0$

Special values:

$$\left\{ {o \atop n} \right\} = \binom{o}{n}, \quad \left\{ {n \atop n} \right\} = 1, \quad \left\{ {n \atop n-1} \right\} = \binom{n}{2}, \quad \left\{ {n \atop o} \right\} = o, \quad \left\{ {n \atop 1} \right\} = 1,$$

$$\left\{ {n \atop 2} \right\} = 2^{n-1}-1 \quad \text{if } n > o.$$

In order to generate the Stirling numbers we will use formula (1) and the special values.

Entry

Function ISTIR has one entry.

L = ISTIR(M,N).

where:	Description
M	is an integer (INPUT)
N	is an integer (INPUT)

The returned value L equals to $\binom{M}{N}$.

Restrictions

The input arguments M,N must be positive integers or zero.

Error Returns

None

Other Subprograms Required

Function ICOMB. This function finds the combinations $\binom{n}{k}$. (see below).

Comments

For a flow-chart and coding of ISTIR see pages B2,C23.

FUNCTION ICOMB

Purpose:

This function finds the combinations $\binom{N}{K}$. These combinations are given by the formula $\binom{N}{K} = \frac{N!}{K!(N-K)!}$ where $N \geq K$.

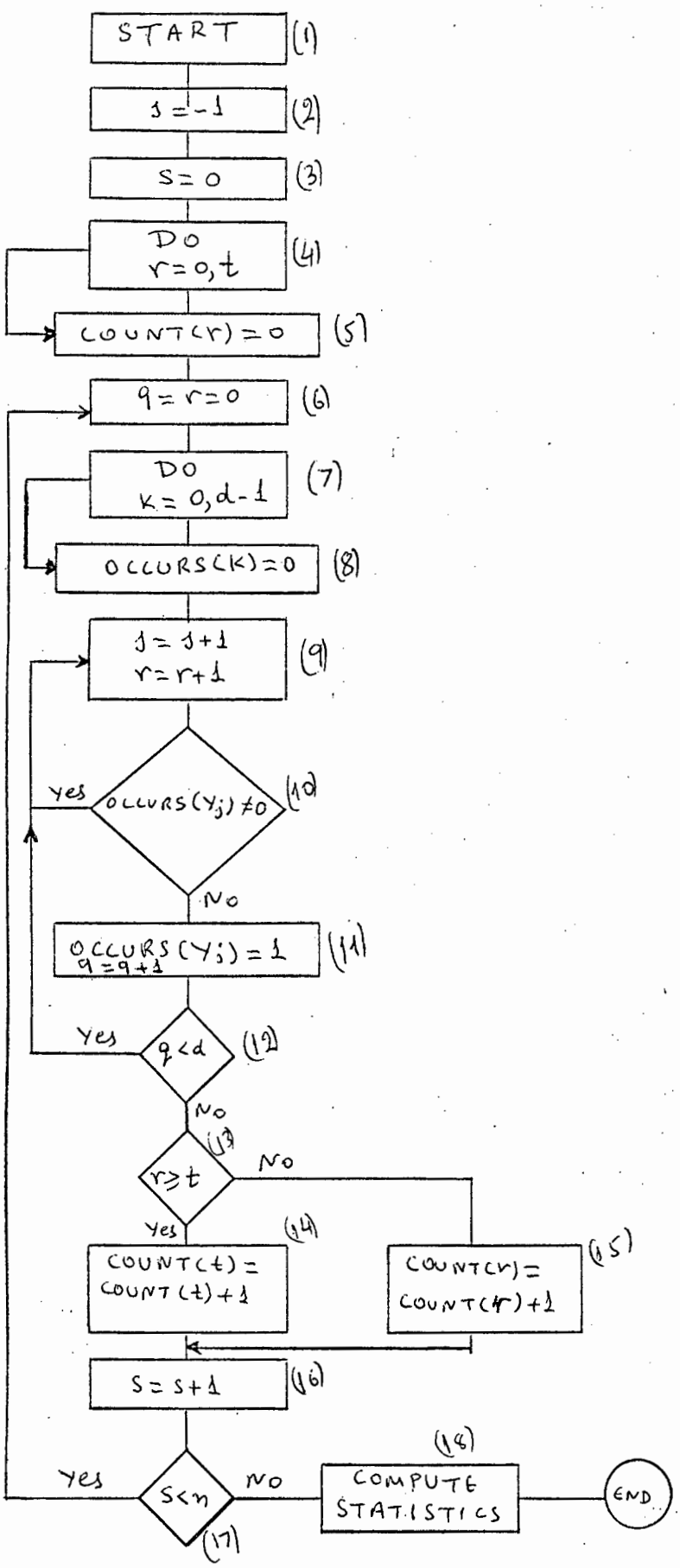


Figure: Coupon Collector's test.

So given a sequence of integers $y_0, y_1, \ldots$ with $0 \leq y_j < d$ this algorithm counts the length of n consecutive segments in order to get a complete set of integers in the range 0, d-1. At the end of the computation COUNT(r) contains the number of segments with length r for $0 \leq r < t$ is a given integer, and COUNT(t) contains the numbers with length $\geq t$. In Boxes 4 and 5 the counter COUNT(r) is set initially equal to zero for $r = 0, \ldots, t$.

In Boxes 7,8 OCCURS(k) is set equal to zero and in box 9 a random number $y_j, 0 \leq y_j < d$ is generated while the index r is increased by one.

In Box 10 if OCCURS(y_j) is zero this means that the value y_j is for the first time generated so OCCURS(y_j) is set equal to one and the variable q is increased by one. The variable q counts the distinct values of y_j. If OCCURS(y_j) $\neq 0$ this means that the value y_j has appeared before so we go to box 9 to generate a new random number.

If q equals to d (Box 12) this means that we have a complete set, so we increase COUNT(r) by 1 if $r < t$ or COUNT(t) by one otherwise.

In Box 17 a test is made to see if $s < n$ and if yes we go to box 6 and the process is repeated again, if no we go to box 18 to compute the chi-square statistic.

The chi-square test is to be applied to COUNT(d), COUNT(d+1),..., COUNT(t) with k = t-d+1 categories after the above algorithm has counted n lengths.

The corresponding probabilities are:

$$p_r = \frac{d!}{d^k} \left\{ {r-1 \atop d-1} \right\}, \quad d \leq r < t, \quad p_t = 1 - \frac{d!}{d^{t-1}} \left\{ {t-1 \atop d} \right\}$$ where the $\{\ \}$ denotes the stirling numbers of the second kind.

A simple example is given:

Let the input sequence 1101221022120202001212201010201121

with d = 3 and n = 7.

The seven subsequences are 11012 (length 5), 210 (length 3) 22120 (length 5), 202001 (length 6), 21220 (length 5) 10102 (length 5) and 0112 (length 4).

The program which puts in action the collector's test is called COLLECT.

Program COLLECT

Purpose:

The input sequence $y_0, y_1, \ldots$ where $0 \leq y_j < d$ is tested for the lengths of the subsequences $y_{j+1}, \ldots y_{j+r}$ required to get a complete set of integers from 0 to d-1. A chi-square test is then applied to the above lengths.

Nomenclature

All variables have been defined in the text.

COLLECT INPUT

COLLECT input data has the following format.

card 1	FORMAT (I3)
col 1-3	NN, is the number of tests performed.
card 2	FORMAT (I6, I4, I5, I12)
col 1-6	N denotes the desired number of lengths of segments, which are required to get a complete set of integers from 0 to d-1.
col 7-10	D is the integer d, as explained in the text.
col 11-15	T is the maximum length of a "coupon collector" segment.
col 16-27	MS is the starting number of the generator IRAND.

COLLECT INPUT DATA

see next page

12			
500	5	8	1
500	5	8	11
500	5	8	1234
500	5	8	2
3000	6	10	4444444
3000	6	10	5643789
3000	6	10	3400000
3000	6	10	111006509
14000	7	12	346777777
14000	7	12	89
14000	7	12	123456
14000	7	12	66665555

Other Subprograms Required

1) IRAND function

2) IPROD function to calculate d factorial (see page 59)

3) ISTIR function, (see page 57)

4) CHI function

Comments

Twelve different sequences of random numbers were tested. The results are presented in page C27.

Four of them are 'almost suspect' namely 90.61%, 93.58%, 90.76%, 91.23% and one 'suspect' namely 1.51%.

The same test was run using MRAND instead of IRAND and the results of this test are presented in page C29.

Two of them are 'suspect' namely 97.41% and 97.17%.

We can say that both generators have almost the same behaviour in respect to this test. A coding of program COLLECT is given on page C26.

3.10 PERMUTATION TEST

In this test we divide the input sequence into n groups of t distinct elements each, that is,

$(V_{jt}, V_{jt+1}, \ldots\ldots, V_{jt+t-1})$, $0 \leq j < n$.

The elements in each group can have t! possible relative orderings; the number of times each ordering appears is counted and a chi-square test is applied with number of categories k = t! and with probability 1/t! for each category.

A convenient way to find which particular relative ordering a group of t elements has, is given below.

Algorithm B.

Given a set of distinct elements $(U_1,U_2,\ldots,U_t)$ we compute an integer $f(U_1,\ldots, U_t)$ such that $0 \leq f(U_1,\ldots, U_t) < t!$ and $f(U_1,\ldots., U_t) = f(V_1,\ldots., V_t)$ iff $(U_1,\ldots., U_t)$ and $(V_1,\ldots., V_t)$ have the same relative ordering.

Step 1

set r = t

Step 2

Find the maximum of $(U_1,U_2,\ldots., U_r)$ and suppose that U_s is the maximum. Set C (r) = s-1.

Step 3.

Exchange $U_r \leftrightarrow U_s$.

Step 4.

r = r-1. If r > 0 return to step 2.

Step 5.

The desired function value is now given by the formula

(1) f = c(t)+tC(t-1)+t(t-1)C(t-2)+ ..... +t!C (1).

From step 2 we observe that $0 \leq C(r) < r$ so in particular C(1) is always zero. Since C(r) may take on r different values we can see that $0 \leq f < t!$

A function which finds which particular relative ordering a group of t elements has, and which is based on the above algorithm is given in the next page.

FUNCTION KF

Purpose:

The value of this function is an integer which uniquely characterizes the initial order of a group of elements.

Entry:

Function KF has one entry:

L = KF(U,IT)

where:	Description
KF	is the name of the function.
U	is an array of real numbers: (INPUT)
IT	is the size of the array U: (INPUT)
L	is the integer value of the function KF, where $0 \leq L < IT!$ which uniquely characterizes the initial order of the group of elements U.

Restrictions

None

Error Returns

None.

Other Subprograms Required

Subroutine MAXIM. This subroutine finds the position of the maximum element in a group $(V_1, \ldots, V_t)$. For further details see supporting information below.

Programming Method:

KF is coded in Fortran V.

Comments

For a program of the function KF see page C29.

Supporting Information

As we have seen in step 2 of the algorithm B it is required to find the position of the maximum element V_s among the elements of an array $(V_1, V_2, \ldots, V_t)$.

Subroutine MAXIM below does it for us.

SUBROUTINE MAXIM

Entry:

MAXIM has one entry:

CALL MAXIM(V,IT,IS)

where:	Description
V	is an array of real numbers, which will be searched to find the position of the maximum element in it. (INPUT).
IT	is the size of the array V. Integer. (INPUT).
IS	is the position of the maximum element of the array V. Integer (OUTPUT).

Restrictions

None

Error Returns

None

Other Subprograms Required

None

Comments

For a flow chart and coding of the subroutine MAXIM see pages B3, C30.

TEST 6

Purpose:

The input seuqence is divided into NG groups of T elements each. The elements in each group can have T! possible relative orderings.

The number of times each ordering appears is counted and a chi-square test is applied with K = T! and probability 1/T! for each ordering.

Programming Method

TEST 6 is coded in Fortran V.

Nomenclature

M	is the number of tests performed.
N	denotes the number of elements of the input sequence.
NG	is the number of groups into which the input sequence is divided.
T	is the number of elements in each group
T NN	is the number of categories.
ORD	is an array which represents the NN categories.
PP	is the probability 1/T!
NF	is the number of degrees of freedom.
VV	is the chi-square statistic.

Other Subprograms Required

1) IPROD function to calculate the T factorial (see page 59)

2) KF function

3) Subroutine MAXIM (see page 66).

Comments

Twelve different sequences were tested. Each of the first four was divided into 100 groups of 3 elements, each of the next four was divided into 400 groups of 4 elements and each of the last four was divided into 5000 groups of 5 elements.

The output is presented on page C32.

Two results are 'almost suspect' namely 7.47%, 91.61% while the remaining 10 tests were passed successfully. For a coding of TEST6 see page C31.

TEST 6 INPUT

TEST 6 input data has the following format.

card 1 FORMAT (I3)

col 1-3 M is the number of tests performed.

card 2 FORMAT (I8,I3,I12)

col 1-8 N is the number of elements of the input sequence.

col 9-11 T is the number of elements in each group.

col 12-24 MS is the starting number of the generator IRAND.

TEST 6 INPUT DATA

```
   12
  300      3            11
  300      3           345
  300      3      34567890
  300      3        123409
 1600      4             1
 1600      4             2
 1600      4            33
 1600      4          1111
25000      5       1443256
25000      5      12300098
25000      5      44445566
25000      5          4456
```

The same test was run using MRAND instead of IRAND. The results of this test are presented on page C33.

We observe that one of them is 'almost suspect' namely 6.47%, and one 'suspect' namely 1.9% while the others were passed successfully. Concluding we can say that IRAND function behaves slightly better than MRAND in respect of this test. Format and input data is exactly the same as above. As initial number for the first generator of MRAND the last number generated by the entry TABLE was used, while for the second generator the input value MS was used.

KOLMOGROROV-SMIRNOV TEST (K-S TEST)

The K-S test like the chi-square test is a non-parametric test used to determine whether an empirical distribution agrees with some hypothesized distribution. The difference between the two tests is that the chi-square is based on class frequencies while the K-S test is based on the cumulative distribution function.

($F(x)$, say where $F(x) = P[X \leq x]$).

Furthermore, strictly speaking the K-S test is only applicable to continuous distributions and where the distribution is completely specified.

Define $F_n(x) = \frac{\text{No. of observations} \leq x}{n}$ where n = total number of observations.

Now suppose that we have n ordered observations $X_1 \leq X_2 \leq \ldots \leq X_n$. It therefore follows that,

$$(1) \quad F_n(x) = \begin{cases} 0 & X < X_1 \\ \frac{j}{n} & X_j \leq X \leq X_{j+1}, \quad j=1,2,\ldots,n-1 \\ 1 & X \leq X_n \end{cases}$$

The test statistics are

$$(2) \quad +K_n = \sqrt{n} \max_x (F_n(x) - F(x))$$

$$(3) \quad -K_n = \sqrt{n} \max_x (F(x) - F_n(x))$$

Here $+K_n$ measures the greatest amount of deviation when F_n is greater than F and $-K_n$ measures the maximum deviation when F_n is less than F. As in the chi-square test we may look up the values $+K_n$ and $-K_n$ in a percentile table to determine if they are significantly high or low. We present a table on page A2 which may be used for this purpose. For example, the probability that $+K_4$ is greater than 0.01943 is 99%. As already indicated the computation of $F_n(x)$ requires the ordering of the observations $X_1, \ldots, X_n$ into ascending order. For a convenient flowchart of this algorithm see page B4.

The point is now, how many observations should be taken in order that the K-S test be performed successfully?

If the random variables X_j belong to the distribution function G(x) while they are assumed to belong to the distribution F(x), it will take a large value of n to prove $G(x) \neq F(x)$; on the other hand large values of n will average out locally non-random behaviour.

A good compromise would be to take n equal to, say, 1000 [11] and to make a fairly large number of calculations of $+K_{1000}$ on different parts of a random sequence, thereby obtaining the values.

(4) $+K_{1000}(1),\ldots,+K_{1000}(r)$.

Our test, which is called TEST 3, was applied to 20 different parts of the random sequence so r=20. We can again apply the K-S test to the above values, and in this case the empirical distribution G(x) of the above values has been found [11] to be equal to $1-e^{-2x^2}$, $x \geq 0$ for $n \to \infty$. So for a large value of n=1000 the distribution function $G_{1000}(x)$ is closely approximated by the expression $1-e^{-2x^2}$. The same remarks apply to $-K_n$, since $+K_n$ and $-K_n$ have the same expected behaviour.

The effectiveness of the K-S test compared with that of the chi-square test is a very difficult problem which is beyond the scope of this report.

Frank J. Massey (AMERICAN STAT. ASSOCIATION JOURNAL, MARCH, 1951) has found some indications that the K-S test is superior to the chi square test. In support of this we shall see in what follows that when MRAND (Univac's 1108 generator) was tested for uniformity in the range [0,1) using the K-S test it highlighted some really bad results while when the same generator was tested for uniformity with the frequency test (see page 38) it passed it with flying colours.

It should be mentioned however that there are cases in which the chi-square test gives more significant results [11].

Program TEST 3

Purpose:

TEST 3, tests the h-pothesis that a sequence of random numbers are uniformly distributed between 0 and 1, using the K-S test with population distribution $F(x) = x$ for $0 \leq x \leq 1$.

Programming Method TEST 3 is coded in Fortran V.

Nomenclature:

XX — is an array which holds the KS values for n = 1000 (see table 2 page A2).

YY — is an array which holds the corresponding 'percentage' values to the above values. (see table 2 page A2).

NTEST — is the number of tests performed.

NSTART — is an array which holds the initial seed for the generator IRAND for each test performed.

X — is an array which holds the 1000 generated real random number between 0 and 1 by IRAND.

Q — is an array which contains the values of the expression of the form $(\frac{j}{n} - F(X_j))$ (see page B4).

Q_1 — is an array which holds the values of the expressions of the form $(F(X_j) - \frac{j-1}{n})$. (see page B4).

RMAX1 — is the maximum of $(\frac{j}{n} - F(X_j))$ $1 \leq j \leq 1000$.

RMAX2 — is the maximum of $(F(X_j) - \frac{j-1}{n})$ $1 \leq j \leq 1000$

KPLUSN — is an array which holds the values $+K_{1000}(1),\ldots,+K_{1000}(20)$

KMINN — is an array which holds the values $-K_{1000}(1),\ldots,-K_{1000}(20)$

YE — is the probability of getting a KS value greater than $+K_{1000}^{(r)}$, $r = 1,\ldots,20$

YE1 — is the probability of getting a KS value greater than $-K_{1000}^{(r)}$, $r = 1,\ldots,20$.

The following variables are used to the output report.

K+20 — is the +KS statistic of the values $+K_{1000}(1),\ldots,+K_{1000}(20)$

K-20 — is the-KS statistic of the values $+K_{1000}(1),\ldots,+K_{1000}(20)$

K+(-20) — is the +KS statistic of the values $-K_{1000}(1),\ldots,-K_{1000}(20)$

K-(-20) — is the -KS statistic of the value, $-K_{1000}(1),\ldots,-K_{1000}(20)$

Other Subprograms Required

IRAND function. It generates integers in the range $[0,2^{35})$, which are subsequently converted into real numbers in the range $[0,1)$

SORTRL subroutine. This subroutine sorts the real numbers $X_1,X_2,\ldots X_n$ into ascending order. For further details see 'supporting information' below.

INTER subroutine. This subroutine outputs the values YE and YEI by interpolation of the table 2. For further details see 'supporting information' below.

AMAX1 this is an intrinsic function which selects the maximum value in a set of n elements.

Comments

20000 numbers were tested. They were divided into twenty different groups using twenty different starting numbers for the generator IRAND. The output results appear in page C35 . Two values out of twenty of KPLUSN are'suspect', namely 96.871% and 4.587%, while only one value of KMINN, namely 91.826% is 'almost suspect'. The values of K+20, K-20, K+(-20), K-(-20), are quite good. For a program see page C34.

TEST 3 INPUT

TEST 3 input data has the following format:

card 1 - card 20	FORMAT (I12)
col 1-12	MS, the starting number of the generator IRAND.

TEST 3 INPUT DATA

see next page

20
2398
129054783
765
99984567
2000000001
9153
2
7
2900
20816
54091
2934
19850043
723
2231974
56
2983330917
4000
666666666
298656

The same program TEST 3 was run using instead of IRAND the generator MRAND. The results appear in page C36 . Six values of KPLUSN out of twenty are 'suspect', namely 95.205%, 95.854%, 3.983%, 95.218%, 1.205% and 6.585% while 3 values of KMINN are 'suspect', namely 95.784%, 97.373%, 1.31%, two 'almost suspect', namely 9k.976%, 94.818% and one 'reject', namely 0.548%. We see that IRAND behaves much better than MRAND in respect of this test.

(MRAND) TEST 3 INPUT

card 1 - card 20	FORMAT(2I2)
col 1-12	MS, the starting number for the first generator of MRAND.
col 13-24	MS1, the starting number for the second generator of MRAND.

(MRAND) TEST INPUT DATA

2338	7777
129054783	234198707
765	333
99984567	762109
2000000001	3000000002
9153	4153
2	3
7	8
2900	3900
20816	30916
54091	64091
2934	3934
19850043	29567849
723	823
2231974	8455566
56	66
2983330917	3333333333
40000	50000
666666666	777777777
298656	656565

Supporting Information

As we have seen the main program TEST 3 uses the subroutine SORTRL which sorts a set of real numbers $X_1,\ldots,X_n$ into ascending order. The technique we consider here for sorting the data is called the method of interchanges. We proceed as follows:

Step 1.

At the Mth stage of the process the head of the list contains the M-1 smallest elements in increasing order.

Step 2

The remaining N-(M-1) elements are searched by successively comparing the M^{th} element with each other element in the list. Anytime an element smaller than the M^{th} is found, this element and the M^{th} are

interchanged. The result after N-M comparisons is to have the next smallest element in the M^{th} position.

Step 3

After N-1 steps the list will be in order of increasing magnitude.

Figure below is a flow chart for this technique.

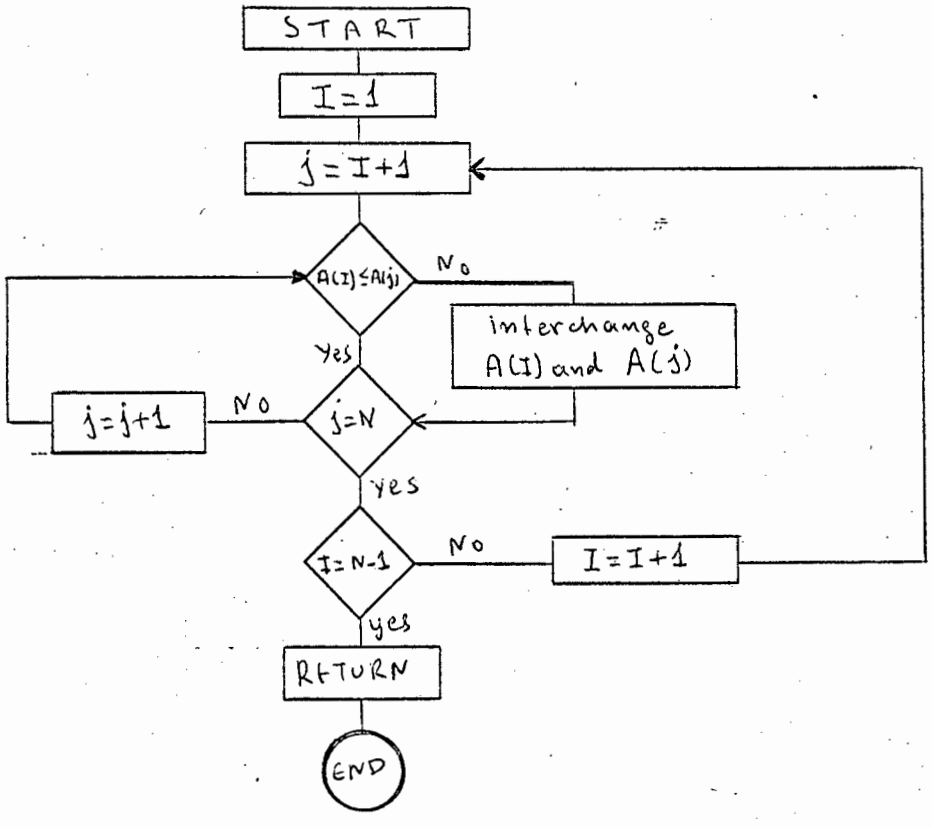


Figure: Sorting by interchanges.

The subroutine which implements the above flow-chart is described below.

Subroutine SORTRL

Purpose:

This subroutine sorts a set of real numbers $X_1,\ldots,X_n$ into ascending order.

Entry:

Subroutine SORTRL has one entry for normal usage:

CALL SORTRL(N,A)

where:	Description
SORTRL	is the name of the subroutine.
A	is the array of the real numbers to be sorted (INPUT,OUTPUT).
N	is the size of the array (INPUT).

Restrictions

None

Other Subprograms Required

None

Error Returns

None

Comments

For a coding of the subroutine SORTRL see page C24.

Subroutine INTER

Mathematical method

As we have seen subroutine INTER is used to interpolate the table 2 (page A2) in order to find intermediate values for the KS value when this lies betweentwo entries of the table, say 99% and 95%. We describe this method mathematically:

Linear interpolation is the process by which values of a function f(x) at points other than the tabulated values (see figure below) $X_1,X_2,\ldots,X_n$ are approximated by using the straight line joining

two adjacent functional values.

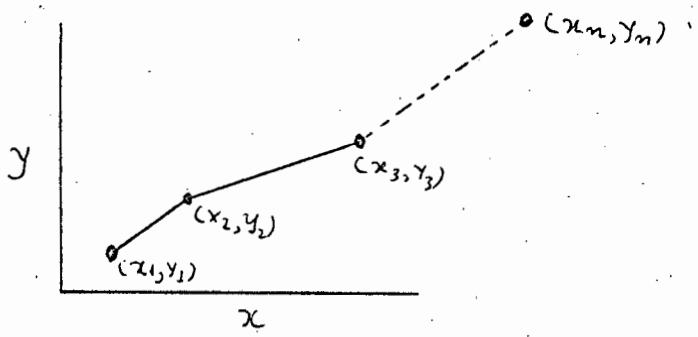


Figure: Linear Interpolation.

The pairs (X_n, Y_n) are given and denote points of the curve f(x). If a point XE where $X_{n-1} < XE < X_n$ is given and we want to calculate the corresponding YE in such a way that the part (XE,YE) belongs to the curve f(X), we apply the following formula:

$$YE = Y_{n-1} + \frac{Y_n - Y_{n-1}}{X_n - X_{n-1}} (XE - X_{n-1})$$

Purpose:

Subroutine INTER interpolates Table 2, (page A2) to find the corresponding probability if the KS values lies between two entries of the table.

Entry

Subroutine INTER has one entry for normal usage:

CALL INTER (X,Y,XE,YE,N,NI,δn,δnI)

where	Description
X and Y	are real arrays of the pairs (X_n, Y_n) which represent points on the curve f(X). (INPUT)
XE	is a real array of points, where $X_{n-1} \leq XE \leq X_n$ for n = 2,3,.....,n. (INPUT)

YE — is a real array of the corresponding interpolated values of the points XE. (OUTPUT).

N — is the size of the array X or Y Integer. (INPUT)

NI — is the size of the array XE or YE Integer (INPUT)

$n — if this formal argument of the sub. INTER and RETURN 7 (see program page C3) is encountered (the first $ is the 7th argument in the list) control transfers to a statement number n in the main program and a diagnostic appears (see below error returns).

$nl — the same applies as above.

Restrictions

The input array X must have been sorted into ascending order.

Error Returns

Two

1. Subroutine INTER checks for X_E values less than smallest X on a curve. If such a value is detected control is transferred into a statement with label n in the main program and an appropriate action is taken. We give an example to clarify the things.

```
      MAIN PROGRAM
           .
           .
           .
      CALL  INTER (X,Y,XE,YE,N,NI,$5,$6)
            GO TO 100
    5       WRITE (5,10)
   10     FORMAT(1X,'ERROR. INPUT VALUE X IS LESS THAN
                 THE SMALLEST X ON CURVE')
            GO TO END
  100       CONTINUE
              .
              .
            END
```

2. Similarly sub. INTER checks for XE larger than the largest X on curve. The same applies as above.

Programming Method

INTER is coded in Fortran V.

Comments

For a flow-chart and program of the sub. INTER see pages B5, C37.

Maximum of t TEST

Let us consider the sequence $U_0, U_2, \ldots, U_{n-1}$, where U_i are uniformly distributed in the range $[0,1)$.

For $0 \leqslant j < n$ let $V_j = \max (U_{tj}, U_{tj+1}, \ldots, U_{tj+t-1})$ (1)

Now the sequence $V_0, V_1, \ldots, V_{n-1}$ (2) has distribution function $F(x) = x^t$ $0 \leqslant x < 1$ and we apply the Kolmogorov-Smirnov test to the sequence (2). We prove now that the sequence (2) has distribution function $F(x) = x^t$.

Let V a random variable, we can define a function $F(x) = P_r (V \leqslant x)$ which is the distribution function of V.

Let us now consider the quantity,

(3) $V = \max(X_1, X_2, \ldots, X_t)$ where $X_1, X_2, \ldots, X_t$ are uniformly distributed variables between 0 and 1.

So,

X_1 has distribution function $F_1(x) = x$

X_2 has distribution function $F_2(x) = x$

$0 \leqslant x < 1$

X_t has distribution function $F_t(x) = x$.

Now $V \leqslant x$ or $\max (X_1, X_2, \ldots, X_t) \leqslant x$ iff $X_1 \leqslant x$ and $X_2 \leqslant x$ and $\ldots X_t \leqslant x$.

But the probability that t independent events occuring simultaneously is the product of the individual probabilities.

So V has distribution function, $F_1(x)F_2(x)\ldots F_t(x)$ or $x.x.\ldots.x = x^t$.

Program TEST II per forms the maximum of t test with t taking the values 2,3,4,5 respectively.

Program TEST II

Purpose:

This program performs the maximum of t test, where t takes on the values 2,3,4,5 respectively.

Nomenclature

The same terminology was used as in TEST 3 (Kolmosporov-Smirnov test).

Other Subprograms Required

1. IRAND function, which generates integer values uniformly distributed in the range $[0, 2^{35})$ which are subsequently converted into real values in the range $[0,1)$.
2. AMAX1 , this is an intrinsic function which chooses the maximum number in a set of t elements.
3. Subroutine INTER. This subroutine interpolates the Kolmogorov-Smirnov table (see page C37).
4. Subroutine SORTRL. This subroutine sorts a set of real numbers $V_1,\ldots,V_n$ into ascending order.

TEST II INPUT

TEST II input data has the following format.

card 1	FORMAT (I4,I8,I3)
col 1-4	M is the number of total tests performed.
col 5-12	N is the number of random numbers examined in each test.
col 13-15	T is the value of t (see text)
card 2-21	FORMAT (I12)
col 1-12	MS, the starting number of IRAND generator.

Comments

The output results are presented on pages C39-C46.

The table on the next page summarizes those results.

TABLE

	IRAND				MRAND			
t	Almost suspect	Suspect	Reject	Acceptable values	Almost suspect	Suspect	Reject	Acceptable values
2	2	3	-	35	2	3	-	35
3	3	2	-	35	4	3	1	32
4	6	4	-	30	1	2	4	33
5	3	1	-	36	3	1	2	34

We observe that in the maximum of 2 test both IRAND and MRAND have the same statistical behaviour, while in the other tests IRAND was proved superior to MRAND generator. For a program of TEST II see page C39.

Serial Correlation Test

The last test examined in this report is the serial correlation test. In chaper I, page 18 we have seen that the correlation coefficient between nearby terms V_i and $V_{(i+1) \bmod n}$ of the linear congruential generator - where V_i are real numbers between 0 and 1 - is given by the formula:

$$(1) \qquad C = \frac{n(V_0V_1+.....+V_{n-2}V_{n-1}+V_{n-1}V_0) - (V_0+V_1+....+V_{n-1})^2}{n(V_0^2+.....+V_{n-1}^2) - (V_0+.....+V_{n-1})^2}$$

We observe from formula (1) that the terms V_i V_{i+1} are not completely independent of V_{i+1} V_{i+2}, so the serial correlation coefficient is not expected to be exactly zero.

Since the exact distribution of C's is not known it has been conjectured [11] that the value of C should be between $\mu_n - 2\sigma_n$ and $\mu_n + 2\sigma_n$, 95% of the time.

μ_n and σ_n are given by the following formulas:

(2) $\mu_n = \frac{-1}{(n-1)}$, $\sigma_n = \frac{1}{(n-1)} \sqrt{\frac{n(n-3)}{n+1}}$, $n > 2$

TEST 9 is the name of the program which examines whether the correlation coefficient C of n uniformly distributed number lies between the above (2) limits.

Program TEST 9

Programming Method

This program is coded in Fortran V.

Nomenclature

MI is the value of the expression μ_n (see formulas (2) in the text).

SI is the value of the expression σ_n.

MD is the left point of the interval $[\mu_n-2\sigma_n, \mu_n+2\sigma_n]$

SD is the right point of the above interval.

NUMER is the numerator of the expression (1) in the text.

DENOM is the denominator of the expression (1) in the text.

PS is the probability the serial correlation coefficient to lie in the range [MD,SD] .

PF is the probability the serial correlation coefficient to lie outside the above range.

C is the correlation coefficient.

Other Subprograms Required

IRAND function.

Comments

Both generators - that is IRAND and MRAND - were tested using the same input data. The output results are presented in page C48. From these results we see that both generators have the same expected behaviour.

TEST 9 INPUT

TEST 9 input data has the following format.

card 1	FORMAT (I4)
col 1-4	M, is the number of tests performed.
cards 2-31	FORMAT (I6,I12)
col 1-6	N, is the number of random numbers examined in each test.
col 7-18	MS, is the starting number of the generator IRAND.

For a program of TEST9 see page C38.

CONCLUSIONS

Thirteen tests were executed to "prove" the hypothesis that the statistical properties of random numbers produced by the generators IRAND and MRAND approximate sufficiently well the statistical properties of numbers generated by an idealized chance device which selects numbers from the interval (a,b) where $0 \leq a,b \leq 2^{35}-1$ independently and with all numbers equally likely. Table 3 of page 85 summarizes the results of the tests performed.

First of all we observe that both generators pass all the tests successfully. Particularly IRAND generator indicates better behaviour in seven tests (frequency, serial, permutation, KS test, maximum of 3, maximum of 4 and maximum of 5) while MRAND generator in four tests (Run gap, Poker, Coupon collector's test). In two other tests (Maximum of 2 and serial correlation test) both generators have the same expected behaviour.

As we have seen the K-S test and the maximum of t tests examine if the cumulative step function of a random sample of N observations is close enough to the theoretical population distribution F(X), which equals to X ($0 \leq X < 1$) for the KS test and X^t ($0 \leq X < 1$) for the maximum

of t test. In these tests the behaviour of IRAND generator is much better than MRAND's. This indicates that the empirical distribution of random samples produced by IRAND generator approximate better the theoretical curves $F(X) = X$ (uniformity), and $F_t(X) = X^t$ $(0 \leq X < 1)$ than those of MRAND generator.

In chapters 4,5,7 we 'test' generator IRAND in a variety of Monte Carlo problems, and as we shall see this generator gives some excellent results. Taking into consideration the speed of the modified generator compared with that of MRAND and the better statistical performance , we believe that this generator can definitely be recommended.

TABLE 3

		IRAND				MRAND				
	TEST	ALMOST SUSPECT	SUSPECT	REJECT	ACCEPTABLE VALUES	ALMOST SUSPECT	SUSPECT	REJECT	ACCEPTABLE VALUES	BETTER GENERATOR
1	Frequency test	1	-	-	9	1	-	1	8	IRAND
2	Serial	2	-	-	10	-	-	1	11	IRAND
3	Run test	3	3	1	32	5	1	-	33	MRAND
4	Gap test	4	4	-	12	2	-	1	17	MRAND
5	Poker test	3	2	1	10	2	4	-	10	MRAND
6	Collector's test	4	1	-	7	-	2	-	10	MRAND
7	Permutation test	2	-	-	10	1	1	-	10	IRAND
8	KS test	1	2	-	37	2	9	1	28	IRAND
9	Maximum of 2 test	2	3	-	35	2	3	-	35	Same behaviour
10	Maximum of 3 test	3	2	-	35	4	3	1	32	IRAND
11	Maximum of 4 test	6	4	-	30	1	2	4	33	IRAND
12	Maximum of 5 test	3	1	-	36	3	1	2	32	IRAND
13	Serial correlation test									Same behaviour

CHAPTER 4.

APPLICATIONS OF THE UNIFORM DEVIATES IN SOLVING MONTE-CARLO PROBLEMS.

4.1 INTRODUCTION.

In the preceding chapter we tested the IRAND generator in a series of statistical tests and it was proved that this generator can be considered as a random device which can output sequences of numbers independent of each other and following the uniform distribution.

The purpose of this chapter is to test those numbers in a variety of practical applications known as Monte-Carlo problems. These are a set of procedures for solving certain types of problems through the use of random numbers.

Perhaps the main reason for doing extensive testing of random number generators is to convince ourselves that these generators can be reliably used as a source of random numbers which are needed by Monte-Carlo problems.

Five Monte-Carlo problems are examined in this chapter. First, a mathematical solution for these problems is derived and subsequently the theoretical results are compared with those which have been found by a simulation process.

As we shall see the simulated results are very close approximations to the theoretical ones.

The first application we shall consider is a special case of the so called "random walk" problem.

4.2 A RANDOM WALK PROBLEM.

A dog is lost in a square maze of corridors. (Fig 1). At each intersection he chooses a direction at random and proceeds to the next intersection, where he again chooses, at random and so on. What is the probability that a dog starting at intersection (i,j) will eventually emerge on the south side?

This is a special case of a random walk in the plane [5]. We suppose that there are 9 interior points as shown in the figure 1. with P(i,j) we symbolize the probability that a dog starting at intersection (i,j) will eventually emerge on the south side. Assuming that at each intersection he reaches, a dog is as likely to choose one direction as another, and having reached any exit the walk is over, probability theory [5] offers the following nine equations for the P(i,j).

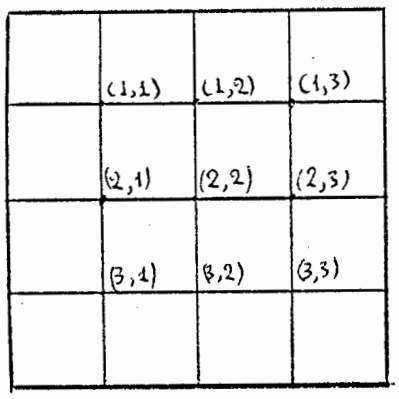


Fig. 1

$$
\begin{aligned}
&4P(1,1) - P(1,2) - P(2,1) = 0\\
&-P(1,1) + 4P(1,2) - P(1,3) - P(2,2) = 0\\
&-P(1,2) + 4P(1,3) - P(2,3) = 0\\
(A)\quad &-P(1,1) + 4P(2,1) - P(2,2) - P(3,1) = 0\\
&-P(1,2) - P(2,1) + 4P(2,2) - P(2,3) - P(3,2) = 0\\
&-P(1,3) - P(2,2) + 4P(2,3) - P(3,3) = 0\\
&-P(2,1) + 4P(3,1) - P(3,2) = 1\\
&-P(2,2) - P(3,1) + 4P(3,2) - P(3,3) = 1\\
&-P(2,3) - P(3,2) + 4P(3,3) = 1
\end{aligned}
$$

First we will try to find the theoretical solution of this system and then to compare these theoretical values with those which we will find simulating the random walk in the plane.

We observe from the above system that the main diagonal coefficient in each row dominates the other coefficients in that row, so the best policy is to attack this problem using the Gauss-Seidel method. This method is described briefly below.

4.3 GAUSS SEIDEL METHOD.

Let us state briefly the method in terms of a system of three simultaneous equations in three unknowns.

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

where

(1) $a_{ii} > \sum_{j \neq i} a_{ij}$, $i = 1,2,3$

Suppose we made guesses at the values of x_2 and x_3, zero will work.

Then we solve the first equation for x_1, writing a prime to indicate that this is a new approximation.

$$x_1' = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}}$$

Now using the new value for x_1 and the initial guess at x_3, we solve the second equation for x_2:

$$x_2' = \frac{b_2 - a_{21}x_1' - a_{23}x_3}{a_{22}}$$

Finally, using the new approximations to x_1 and x_2, we solve the third equation for x_3:

$$x_3^{1} = \frac{b_3 - a_{31}x_1^{1} - a_{32}x_2^{1}}{a_{33}}$$

This process of computing a new value for each of the variables constitutes one interation.

We now perform another iteration, always using the most recently computed value of each variable. Since condition (I) holds the approximations will converge to a solution of the system [16], regardless of the initial guesses used.

On page 92 a convenient flow-chart of this method is presented.

Box 2 clears the array for the unknown x putting them equal to zero. (Initial guesses for the unknowns).

Box 3 contains the input to the subroutine. This input includes the number of the equations n, and test for convergence ε to decide when the approximations are close enough, and an iteration counter to stop the process if for any reason it failes to converge.

MAX denotes the maximum number of the iterations to be permitted. The coefficients a_{ij} and the constant terms are included in the input of the subroutine as well.

The connector I, takes us to the part of the flow-chart that begins an iteration. If the maximum absolute difference between the last two approximations to any variable is less than ε, the iteration scheme will be stopped. The variable that represents the largest difference found so far is called BIG.

The line i=1 represents part of a DO statement action.

The SUM=0 box is the start of the computation of a new approximation to one variable.

This involves accumulating a sum of products ($a_{ij}x_j$) in SUM, which we accordingly set to zero.

The decision box i=1 forms the sum of the off diagonal terms in one row.

Except for the first and last equations, all other equations have some terms before the diagonal and some after. Here we use two DO loops, one running from 1 to i-1, the other from i+1 to n.

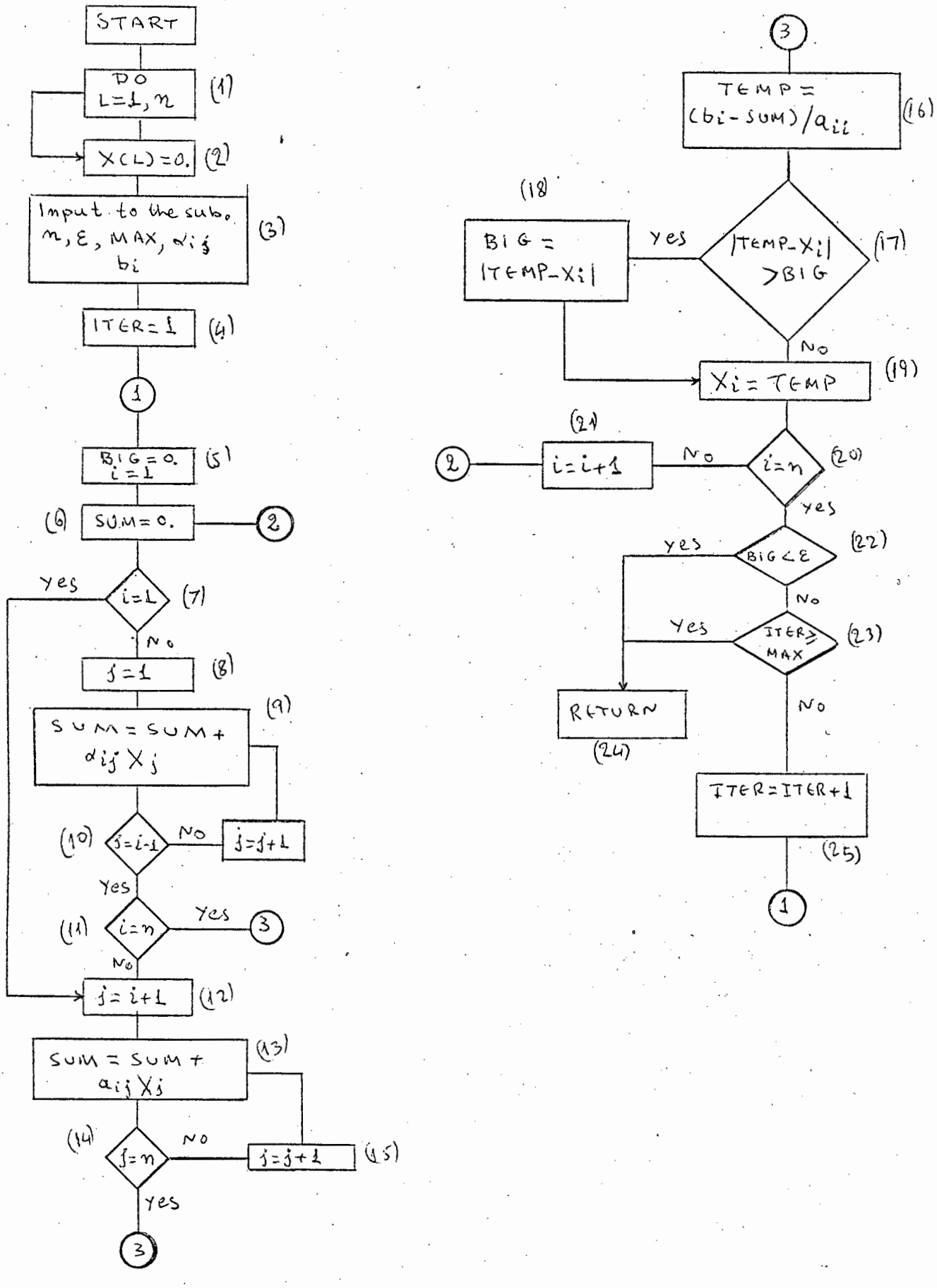
The first DO loop is skipped for equation 1 and the second for equation n.

At connector 3 we set TEMP equal to the new approximation, and we ask whether the absolute value of the difference between the new and old values is greater than BIG and, if so, set BIG equal to the difference.

The temporary storage location TEMP keeps the new approximation and we go back to connector 2 if more equations remain.

If i=n at this point an iteration has been completed.

If BIG is less than ε the iteration scheme has finished, and the unknowns are returned to the main program. Otherwise we ask whether the iteration counter exceeds the MAX value and either return the values of the knowns to the main program or increment the counter by 1 and go back at connector 1.



Flowchart: Gauss-Seidel method

SUB-ROUTINE GAUSS.

Purpose.

This sub-routine finds the solution of n linear simultaneous equations, using the GAUSS-SEIDEL method.

Entry.

GAUSS has one entry.

CALL GAUSS(A,B,N,EPS,MAXIM,X,ITER)

where	Description	Type
A	is an array which contains the coefficients of the equations.	Real INPUT.
B	is an array which contains the constant terms of the equations.	Real INPUT.
N	is the size of the array A or B.	Integer INPUT.
EPS	is the convergence criterior.	Real INPUT.
MAXIM	is the maximum number of the interations desired.	Integer INPUT.
X	is the solution vector.	Real OUTPUT.
ITER	is the actual number of the interations performed at the end of the computation.	Integer OUTPUT.

Comments.

For a program of the sub GAUSS see page C49.

The main program which utilizes the above sub-routine in order to find the solution vector of the system (I) on page C25 is described below.

PROGRAM MAIN I.

Purpose.

This program solves the system of the nine linear equations on page 88.

Programming method.

MAIN I is coded in Fortran V.

Comments.

Our task consists of making the coefficients of the unknowns and the constant terms of the system (I) (Page 89) available to the sub-routine GAUSS.

One way is to punch all this data onto cards. Because of the fact that only the numbers 4, -1,0 are involved an alternative procedure can be illustrated (see program MAIN I, page C24).
The output results are, (see page C50).

(A)
$P(1,1) = x_1 = 0.07143$
$P(1,2) = x_2 = 0.09821$
$P(1,3) = x_3 = 0.07143$
$P(2,1) = x_4 = 0.18750$
$P(2,2) = x_5 = 0.25$
$P(2,3) = x_6 = 0.18750$
$P(3,1) = x_7 = 0.42857$
$P(3,2) = x_8 = 0.52679$
$P(3,3) = x_9 = 0.42857$

Our present intention, however, is to solve this problem using the random number generator IRAND.

The following program carries out this task.

PROGRAM WALK.

Purpose.

WALK simulates dog's random walk in the plane.

Programming method.

WALK is coded in Fortran V.

General description of the method.

Choosing one of the interior points of the maze (see fig.1 page 88), as the starting point, a random number is to determine which direction is taken. The new position of the dog must be then recorded, after which similar steps follow until a boundary position is reached. The walk just finished is then counted as either a success or failure - depending on which side of the maze has been reached - and another walk is begun. After a number of such trials - in our program 30 000 - the ratio of successes to total trials may be taken as the probability P(I,I), that is, to emerge the dog from the south side of the maze starting his walk from the position (i,j). A different starting point may then be chosen and the whole process is repeated.

Nomenclature.

KK is set equal to 2^{33}. (One fourth of the computer size word).

(INITI,INITJ) The initial point of the plane, from which the dog starts his walk.

NUMSUC This location contains the number of successes when a random walk is over (that is, when the dog reaches the south boundary of the maze).

NUM is a random number in the range $[0,2^{35})$ which is generated by IRAND.

INDEX equals to (NUM/KK) + 1, that is, one of the integers 1 or 2 or 3 or 4. These integers are interpretated as directions N,E,S,W respectively.

Comments.

When this program was executed, the following values of the probability P were found. (See page C52 for the output results).

P(1,1) = 0.0715

P(1,2) = 0.0974

P(1,3) = 0.07

(B) P(2,1) = 0.187

P(2,2) = 0.2491

P(2,3) = 0.1848

P(3,1) = 0,4299

P(3,2) = 0,5299

P(3,3) = 0.4262

We can easily check that the values (B) are very close to the theoretical ones (A).

4.3 NUMBER OF PRIMES IN THE RANGE $[0,2^{35})$.

According to the celebrated "Prime Number Theorem" [8], the number of primes less than x is asymptotically x/lnx (1) and tends to be a little higher than this value.

For example, if x=1000, the number of primes <x is 168 and the value given by the formula (1) is 145.

We will try now to find the approximate number of primes in the range $[0,2^{35})$, that is, when $x=2^{35}$ using sampling techniques.

Let us first find the theoretical values which are given by the formula (1) when $x=2^{35}$. The number of primes, in this case, is found to be 1416303520 (2) approximately and if we wish to find the probability a number to be prime in the range $[0,2^{35})$, we simply divide the above value (2) by 2^{35}. This probability is equal to 0.0412 (3) approximately.

Now in order to find the probability of a number to be prime in the above range using sampling techniques, we think as follows:

We generate twenty different samples of integers in the range $[0,2^{35})$ and in each sample we determine the number of primes. This number is divided by the sample size, and the proportion of a number to be prime in this sample is found. We add the proportions together, and the sum is divided by 20, thus getting the "population" probability of any number to be prime. Population here, is considered the set of integers in the range $[0,2^{35})$.

If we multiply the population probability by 2^{35} we find the approximate number of primes in the above range.

Details of the output of the program which carries out this technique are given on page C54.

The population probability is found to be 0.045 and the number of primes 1541034160. Comparing these results with the above theoretical ones, we note that the discrepancies are due to the fact that the number of samples and the size of them (size 1000) is too small.

We did not take a larger number of samples with greater size because the computer time required to calculate these prime numbers is very long. The execution time of the program which gave the above results was 33 minutes and 39 seconds.

Program PRIMEPROG.

Purpose:

This program was designed to find the approximate number of primes in the range $[0,2^{35})$.

Nomenclative.

NTEST denotes the number of samples.

ISUMP This location holds the number of primes in each sample.

SAMSIZ is the size of the sample.

PP is the probability of a number to be prime in a certain sample.

POPPR is the population probability of a number to be prime.

Other subprograms required.

1. IRAND function, to generate the random samples in the range $[0,2^{35})$.

2. IPRIME function, to determine if a given integer is prime or not (see supporting information).

PRIMEPROG INPUT.

PRIMEPROG input data has the following format:

Card 1. FORMAT(I4,I6)

col 1-4. NTEST is the number of samples.

col 5-10. SAMSIZ is the size of the sample.

Card 2-22. FORMAT(I2).

col 1-12. MS, is the starting number for the generator IRAND.

PRIMEPROG INPUT DATA.

```
20     1000
         91
         71
         90
        341
      90174
    5528814
      54600
       1898
       1073
  301755111
         92
       4948
 2211333999
 9165325656
         61
        668
         38
   87561000
 9991123877
 1234567891
```

Supporting information.

Program PRIMEPROG calls function IPRIME, which determines whether a given integer is prime or not.

Subprogram IPRIME has one entry and the calling sequence is:

L=IPRIME(N),

where N is the input integer. The returned value L will be 1 if N is prime or 0 if N is not prime.

The general procedure on which this function is based is the following:

We divide the input number N by the numbers 2,3,5,7,11,13,17,19,23,..., $\sqrt{N}$ (1) and we check the remainder in each division. If we find a zero remainder for some number of the above sequence, the algorithm terminates and the number N is not prime, otherwise the number N is prime.

We observe from sequence (1) that after the first three terms we alternatively add 2 and 4. This is done to save computer time, since the above sequence does not contain the numbers which are multiples of 2 or 3.

For a flowchart and program of the function IPRIME see pages B6, C55.

4.4 CESARO'S THEOREM.

Cesaro proved in 1881 the following theorem.

Theorem:

If u and v are integers chosen at random, the probability that gcd(u,v)=1 is $6/\pi^2$ or approximately 0.60793.

With the symbol gcd we denote the greatest common divisor of two integers.

We find an approximate solution to this theorem using the random

number generator IRAND.

Ten thousand pairs of random integers are generated which lie in the range $[0,2^{35})$. For each pair the greatest common divisor is calculated and the number of pairs which are relatively prime is found. We divide the found number of relatively prime numbers by 10000 and we compute the approximate probability a random pair of integers to have gcd the unity.

Details of the output and coding of this program can be found on pages C51, C56.

The found probability $\simeq 0.6033$, is close enough to the theoretical one.

The only subprograms required for this program are IRAND generator and IGCD function, which calculates the greatest common divisor of two integers.

This function is based on the following algorithm:

Euclidean algorithm.

This algorithm finds the greatest common divisor, given two non-negative integers u,v, where u>v.

Step 1.

If v=0 the algorithm terminates with output u.

Step 2.

Set r=u mod v, u=v, v=r and return to step 1.

Function IGCD puts in action the above algorithm.

Function IGCD.

Entry.

IGCD has one entry:

L=IGCD(U,V).

where	Description
U	is a negative or zero or positive interger (INPUT).
V	the same as U.

The returned value L is the gcd of U and V.

Restrictions.

None.

Error returns.

None.

Comments.

If the input parameters are negative integers or at least one is negative, a test is made (see program on pageC57) and the sign of the integer is changed so eventually the gcd of |u| and |v| is found.

The first argument need not be greater than the second one. If u<v the function IGCD changes the order of u and v so always the greater of the two is divided by the smaller.

The function IGCD calls the intrinsic function MOD which finds the remainder of two positive integers u,v.

4.5 NUMERICAL INTEGRATION.

We compute the area of the first quandrant of a unit circle. This numerical integration problem will serve to illustrate the use of the Monte-Carlo method in solving a completely deterministic problem.

Any pair of uniform random numbers r_1, r_2 (where $0 \leq r_1, r_2 < 1$) defined over the unit interval corresponds to a point within the unit square of figure 2 and the points satisfying the equation $r_1^2 + r_2^2 = 1$ lie on the circle.

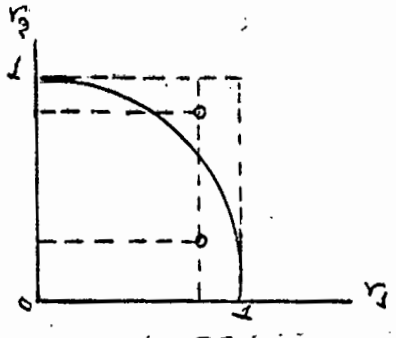


Figure 2

Numerical integration

Let $g(r_1) = \sqrt{1-r_1^2}$. If $g(r_1^0) \leq r_2^0$, the point (r_1^0, r_2^0) lies above the curve. Now accepting and counting the first type of random occurrences and dividing this count by the total number of pairs generated, we obtain a ratio corresponding to the proportion of the area of the unit square lying under the curve.

This ratio will approach to the area of the circle as the number of random pairs increases.

The area of the circle is found by solving the definite integral,

$\int_0^1 \sqrt{1-x^2}\, dx = \frac{x}{2}\sqrt{1-x^2} + \frac{1}{2} \text{ are } \sin x \Big]_0^1 = \frac{\pi}{4} = 0.78539.$

The program given on page C58 is selfexplanatory. The output of this program is AREA=0.7886 which is very close to the above theoretical value.

CHAPTER 5

INTRODUCTION

This chapter examines two useful applications of random numbers. Random sampling and random shuffling. Two problems are solved, using random sampling techniques for the first, while for the second random shuffling is needed. The results compared with the theoretical ones are again extremely good.

5.1 RANDOM SAMPLING

One of the most useful applications of random numbers is an unbiased choice of n records at random from a file containing N records. This problem arises for example in some statistical applications where sampling is needed. In order to select n records at random from a set of N, where $0 < n \leq N$ the following algorithm (ref 1) may be used.

Algorithm: Sampling

Step 1

Set $t = 0$, $m = 0$.

Step 2

Generate a random number U, uniformly distributed between zero and one.

Step 3

If $(N-t)U \geq n - m$ go to step 5. (m denotes the number of records which have been already selected).

Step 4

Select the next record from the sample, and set $m = m+1$, $t = t+1$. If $m < n$ go to step 2; otherwise the sample is complete and the algorithm terminates.

Step 5

Skip the next record (do not include it in the sample), increase t by 1 and go to step 2.

From the above algorithm we observe that at most n records are selected, since the algorithm terminates when $m = n$, (step 4). At the other hand, at least n records are selected, since step 3 will never go to step 5 when the number of records left to be examined is equal to n-m.

So we conclude that exactly n records will be selected.

The more obvious approach for selecting n records out of N is to pick each record with probability n/N. From step 3 we observe that only the first record is selected with probability n/N. The $(t+1)^{th}$ record is picked up with probability $\frac{n-m}{N-t}$ where m are the records which have already been selected.

This apparent contradiction is due to the fact that the probability of selecting the (t+1)th record is an "aposteriori" probability.

We explain this more clearly. The quantity m depends randomly on the selection which took place among the first t elements: if we take the average overall possible choices which could have occurred among these elements, we will find (n-m)/(N-t) is exactly n/N on the average.

For example, consider the second element; if the first element was selected in the sample (this happens with probability n/N), the second element is selected with probability $\frac{n-1}{N-1}$; If the first element was not selected, the second is selected with probability $\frac{n}{N-1}$. The overall probability of selecting the second element is $\frac{n}{N} \cdot \frac{n-1}{N-1} + (1-\frac{n}{N}) \cdot \frac{n}{N-1} = \frac{n}{N}$.

So eventually each element of the file is selected with probability n/N.

Subroutine SAMPLE

Purpose:

Subroutine SAMPLE selects n records at random from a set of N, where $0 < n \leq N$. This sub. is based on the above algorithm.

Entry:

SAMPLE has two entries:

1) The first is for normal usage;
2) Subroutine SAMPLE calls subroutine PINAX which initializes the table from which random numbers in the range $(0,2^{35})$ are generated (see page 62). In order to avoid reinitialization of the table for each subsequent reference to sub. SAMPLE, a second entry point is created.

Standard Entry:

CALL SAMPLE (X,Y,N,L,MS)

where	Description	Type
X	is an array containing N records.	Integer (INPUT)
y	is the output random sample.	Integer (INPUT)
N	is the size of the array X.	Integer (INPUT)
L	is the size of the array y	Integer (INPUT)
MS	is the starting number to the function IRAND which is contained in the subroutine SAMPLE. At the end of the computation MS contains the last number generated by IRAND.	Integer (INPUT,OUTPUT)

Restrictions

L must be less or equal to N.

Error Returns

None

Secondary Entry:

CALL QUICKS (X,Y,N,L,MS)

The arguments of this entry have the same meaning as above.

Special Considerations

The output sample depends only on the starting value MS. Different values of MS will result in different output samples. MS should be chosen in the range $[0,2^{35})$.

Other Subprograms Required

IRAND function to generate random numbers in the range $[0,2^{35})$ which are subsequently converted into real numbers uniformly distributed between zero and one.

Comments

For a flow-chart and coding of the subroutine SAMPLE see pages B7, C64.

5.2 RANDOM SHUFFLING

The sampling problem as previously described can be regarded as the computation of a random combination according to the definition of combination of N things taken n at a time. Let us consider now the problem of computing a random permutation of n objects. We shall call this the "shuffling" problem, since shuffling a deck of 52 cards is nothing more than arranging the cards to a random permutation. We give now an algorithm which shuffles a set of t numbers and which is due to R. Durstenfeld, (CACM 7 (1964) page 420).

Algorithm: Shuffling

Let $X_1, X_2, \ldots, X_t$ be a set of t numbers to be shuffled.

Step 1

Set $j = t$.

Step 2

Generate a random number V, uniformly distributed between zero and one.

Step 3

Set $K = \lfloor jV \rfloor + 1$. (Now K is a random integer between 1 and j).

Exchange $X_K \leftrightarrow X_j$.

Step 4

Decrease j by 1. If $j > 1$ return to step 2 otherwise the algorithm terminates.

A subroutine which puts in action the above algorithm is described below.

Subroutine SHUFFL

Purpose:

Subroutine SHUFFL shuffles a set of n numbers $X_1, X_2, \ldots, X_n$.

Entry:

SHUFFL has two entries

1) The first is for normal usage.

2) The second is exactly the same as in SAMPLE subroutine (see page C62).

Standard Entry:

CALL SHUFFL (X,N,MS)

Where	Description	Type
X	is an array of N numbers to be shuffled.	Integer (INPUT,OUTPUT)
N	is the size of the array X.	Integer (INPUT)
MS	is the starting number of the subprogram IRAND which is contained in the sub. SHUFFL. At the end of the computation MS contains the last number generated by IRAND.	Integer (INPUT,OUTPUT)

Restrictions

None.

Error Returns

None

Secondary Entry

CALL PERMUT (X,N,MS)

The arguments of this entry point have been explained above.

Special Considerations

The same as in SAMPLE subroutine.

Other Subprograms Required

IRAND function.

Comments

For a flow-chart and coding of the subroutine SHUFFL see pages B7, C62.

5.3 A STATISTICAL PROBLEM

An interesting application of the random sampling theory is the following statistical problem.

Table I

Heights	Deviation d = X-A	Frequency (f)	fd	X^2	fX^2
61	-6	5	-30	3721	18605
64	-3	18	-54	4096	73728
A→ 67	0	42	0	4489	188538
70	3	27	81	4900	132300
73	6	08	48	5329	42632
		$N=\Sigma f=$ $=100$	$\Sigma fd=$ 45		$\Sigma fX^2 =$ 455803

Consider that a university class consists of a hundred students. The heights and the corresponding frequencies of these students are given in Table I. In order to find the arithmetic mean of the heights of the students we use the formula.

$$\mu_X = A + \left(\frac{\Sigma fd}{N}\right) = 67 + \frac{45}{100} = 67.45 \text{ inches}$$

The standard deviation is given by the formula,

$$\sigma_X = \sqrt{\frac{\Sigma fX^2}{N} - \left(\frac{\Sigma fX}{N}\right)^2} = 2.92 \text{ inches}$$

Let us suppose that 60 samples of 6 students each are selected (with replacement) from the above table.

The theoretical mean of the sampling distribution of means, given by $\mu_{\bar{X}}$ should equal to population mean 67.45 inches.

(3) So $\mu_{\bar{X}} = 67.45$

The theoretical standard deviation of the sampling distribution of

means, given by $\sigma_{\bar{X}}$ should equal to $\sigma/\sqrt{N}$ where σ = 2.92 from relation (2) and N = 6. So (4) $\sigma_{\bar{X}}$ = 1.191.

Program STAT below is designed to find the mean and standard deviation of the sampling distribution of means by drawing 60 samples of size 6 from the population of the 100 students. Then these results are compared with the above theoretical ones. We use the digits 0,1,2...,99 to number each of the 100 students. Thus the 5 students with height 61 inches are numbered 0-4, the 18 students with height 64 are numbered 5-22 etc. Program STAT is described below.

Program STAT

Nomenclature:

X	As we have seen each student has a number assigned to him. X is an array which holds the student numbers.
N	is the size of the array X.
Y	is an array holding the sample numbers of each student which are drawn from the population of the 100 students.
L	is the size of the array Y (sample size).
H	is the array of heights.
MEAN	is the mean of the heights of the sample.
SUM	is the mean of the sampling distribution of means.
SDTV	is the standard deviation of the sampling distribution of means.

Other Subprograms Required

Subroutine SAMPLE (see page C62) which draws the random samples.

STAT INPUT

STAT input data has the following format.

card 1	FORMAT (2I5,I12)
col 1-5	N is the population size.
col 6-10	L is the sample size.
col 11-22	MS is the argument of subroutine SAMPLE.

card 2 FORMAT (5F4.1)

col 1-4 H, is an array which holds the student heights.

STAT INPUT DATA

```
100        6          6400812
 61.0 64.0 67.0 70.0 73.0
```

Comments

STAT calls SAMPLE at the beginning of the program in order to initialize the PINAX sub. which is incorporated in the subroutine SAMPLE and to output the first random sample. Subsequent references to subroutine SAMPLE are made through its entry point QUICK (see SAMPLE page C62) and this is done to avoid reinitilization of the sub PINAX.
The found values of mean and standard deviation are respectively 67.42 and 1.19 which are very close to theoretical ones. For a program and output results see pages C63, C64.

5.4 THE GAME OF THIRTEEN

An interesting application of the "shuffling" theory is the following game.

Let us consider two players A and B.

A deck of 13 cards of the same colour is thoroughly shuffled. One of them starts throwing the cards one at a time. If at the i^{th} "throw" where $1 \leq i \leq 13$ a card happens to have the number i on its face, the game is considered won by A, otherwise the game is won by B.

Find the probability of the game to be won by A.

For n cards - numbered from 1 to n - the probability theory [2] gives us the answer.

The probability p_A (the game to be won by A) =

$$= 1 - \frac{1}{1.2} + \frac{1}{1.2.3} - \ldots\ldots + (-1)^{n-1} \frac{1}{1.2. \ldots n} \quad (1)$$

and $p_B = 1 - \frac{1}{1} + \frac{1}{1.2} - \frac{1}{1.2.3} + \ldots\ldots + (-1)^n \cdot \frac{1}{1.2\ldots.n}$ (2).

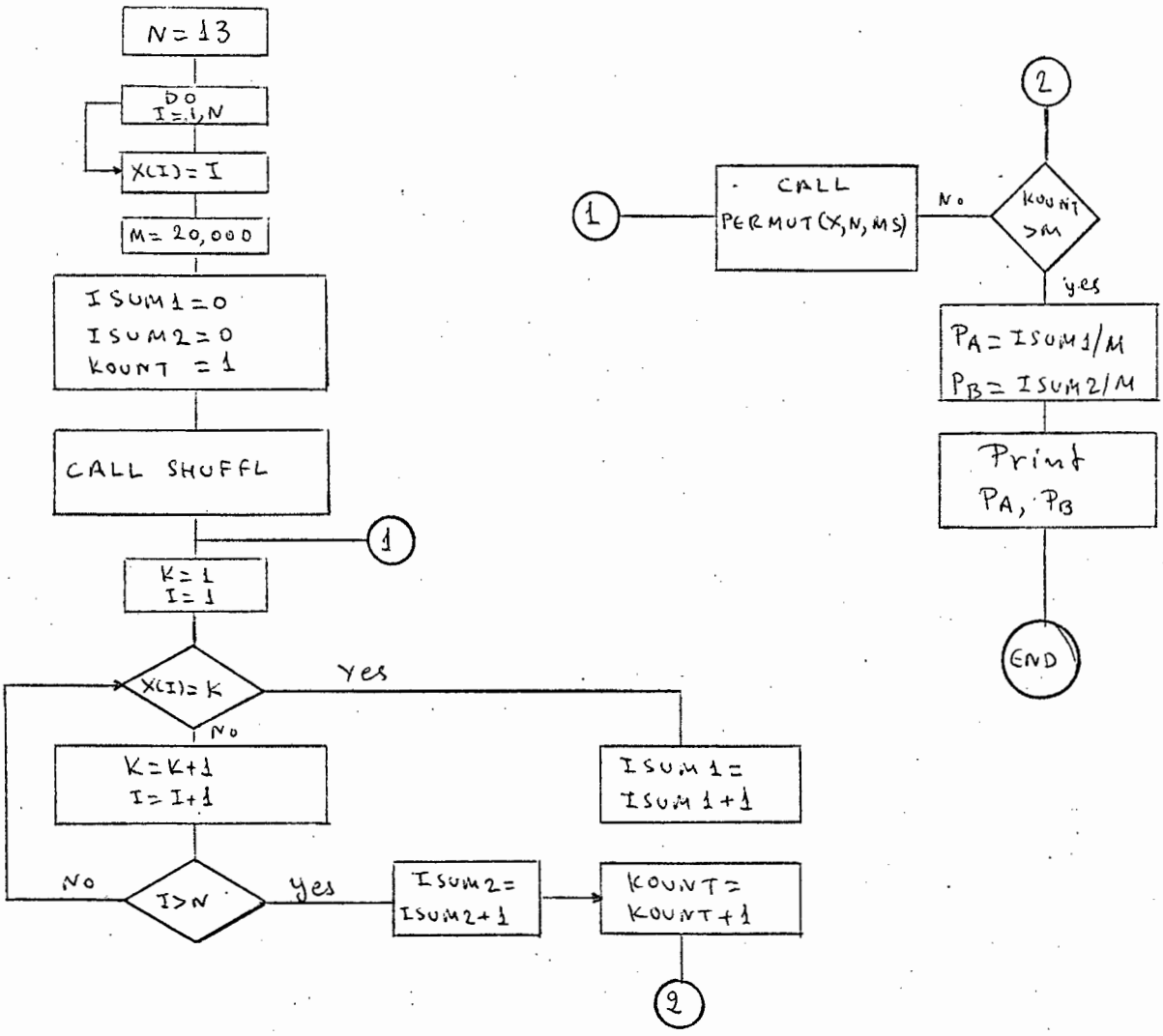
When $n \to \infty$ then we have $p_A = 1 - e^{-1}$ and $p_B = e^{-1}$ or

$p_A = 0.632120055$ .... (3) and $p_B = 0.36787944$ ..... (4).

Since series (1) and (2) converge rapidly for relatively small values of n to the corresponding values (3) and (4), it has been found [2] that for n = 13, $p_A = 0.632120559$ (5) and $p_B = 0.367879441$ (6) approximately.

Our intention is to play this game on a computer using subroutine SHUFFL (see page C62) to shuffle the deck of 13 cards. The game is played 20000 times and the corresponding probabilities are found. The simulated value of p_A is found to be 0.6334 and for p_B 0.3666 which are close enough to the theoretical values (5) and (6). Details of the output and coding of the program can be found on pages C50, C25. The flow-chart on which the program - called MEET - was based is on the next page. This flow-chart is very easy to follow. The only thing we want to add is, that the variable ISUM 1 keeps track of the successes of player A and ISUM 2 of the successes of player B.

The corresponding probability for player A is found (Box 16) by dividing ISUM 1 by M and for player B by dividing ISUM 2 by M.



Flowchart: The game of thirteen.

CHAPTER 6

6.1 OTHER TYPES OF RANDOM QUANTITIES

The general requirement in simulation is for a sequence of random numbers drawn from a distribution that is continuous and non-uniform.

The most common method of deriving such numbers is from a uniformly distributed sequence of random numbers. A real-valued distribution may be expressed in terms of its " cumulative distribution function" $F(x)$. This function can be stated mathematically as follows:

(1) $F(x) = Pr(X \leq x) = \int^{x} f(t)\,dt$. $F(x)$ is defined over the range $0 \leq F(x) \leq 1$ and always increased monotonically from zero to one.

If this function is continuous and strictly increasing then it takes on all values in the range (0,1) and there is an inverse function. $F^{-1}(y)$ such that if $0 \leq y \leq 1$,

(2) $y = F(x)$ iff $x = F^{-1}(y)$.

The function $f(x) = \frac{dF(x)}{dx}$ is called a probability density function (provided that $F(x)$ is continuous).

A general way to compute a random quantity X with the continuous, strictly increasing cumulative distribution $F(x)$ is to set $X = F^{-1}(V)$ (3), where V is a uniformly distributed random number in the range (0,1).

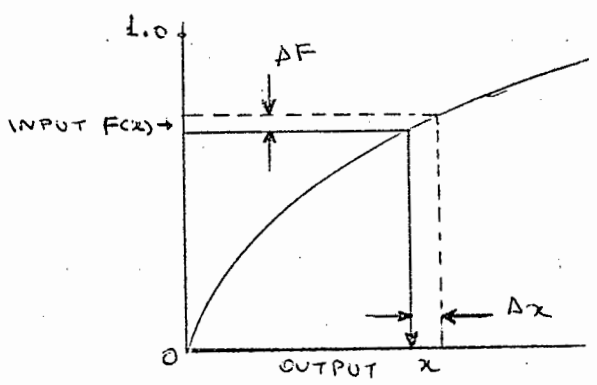


Figure 6.1. Generation of non-uniform continuos random numbers.

For $P_r(X \leq x) = P_r(F^{-1}(V) \leq x) =$

$= P_r(V \leq F(x)) = F(x)$ since V is uniformly distributed between zero and one.

To demonstrate the validity of the result (3) we give an illustrated example: In figure 6.1 the curve represents the cumulative distribution function of the variable X.

Consider now a small interval Δx on the x-axis between x and $x + \Delta x$ and the interval ΔF defined by the corresponding points $F(x)$, $F(x + \Delta x)$. For a number of the output sequence to fall in the interval Δx, the input number must fall in the interval ΔF.

Suppose now that the input numbers are uniformly distributed between 0,1, then the probability of an input number falling in the interval ΔF equals to ΔF. The average value of the probability density function for x over the interval Δx is therefore $\Delta F/\Delta x$.

If now Δx tends to zero, we have $\lim_{\Delta x \to 0} \frac{\Delta F}{\Delta x} = \frac{dF}{dX} = f(x)$

which is by definition the probability density function of x.

This method which generates random variables x having cumulative distribution $F(x)$ is called the inverse transformation method.

Unfortunately, for many probability distributions it is either impossible or extremely difficult to express x in terms of the inverse transformation function $F^{-1}(V)$. In this case we use the following method.

6.2 THE REJECTION METHOD

If the probability function $F(x)$ of a variable X is bounded and x has a finite range say $a \leq x \leq b$, the following technique can be used to generate random quantities having probability function $F(x)$.

This technique requires the following steps:

1) Normalize the range of F by a scale factor C, such that

$$C\,F(x) \leq 1, \quad a \leq x \leq b$$

2) Define x as a linear function of V (V is uniformly distributed between 0,1) that is

$$X = a + (b-a)V \quad .$$

3) Generate pairs of random numbers (V_1, V_2) uniformly distributed in the range $[0,1)$.

4) Whenever we encounter a pair of random numbers that satisfies the relationship

$$V_2 \leq C.f[a + (b-a)V_1]$$

then accept the pair and use $X = a+(b-a)V_1$ as a generated random variable.

These variables will have as probability function the function $F(x)$.

Proof

We have already seen that the cumulative distribution function is defined as

$F(x) = P_r(X \leq x)$, and if X is uniform variate we have

$F(x) = P_r(X \leq x) = x$, $x \in (0,1)$.

Now this method can be easily proved.

We have $P_r(V \leq CF(x)) = C.F(x)$, since V is uniform variable in the range (0,1). Consequently if x is chosen at random for the range (a,b) according to step (2) and then rejected if $V > C.F(x)$ the probability density function of the accepted x's will be $F(x)$.

An application of this method was given in page 102 where we computed the area of the first quandrant of a unit circle.

6.3 CONTINUOUS PROBABILITY DISTRIBUTIONS. NORMAL DISTRIBUTION.

The normal distribution is the best known and most frequently used probability distribution. There are at least two explanations that seem to support its popularity.

"Mathematical proof tells us that, under certain qualifying conditions, we are justified in expecting a normal distribution, while statistical experience shows that, in fact, distributions are often approximately normal." [2].

If a random variable X has a density function f(x) given as follows:

$$(1) \qquad f(x) = \frac{1}{\sigma_X \sqrt{2\pi}} \, e^{-\frac{1}{2}\left(\frac{X-\mu_X}{\sigma_X}\right)^2}, \quad -\infty < X < +\infty$$

then X is said to have a normal distribution with mean μ_X and standard deviation σ_X. If $\mu_X = 0$ and $\sigma_X = 1$ the distribution function is known as the standard normal distribution with density function given by

$$(2) \qquad f(z) = \frac{1}{\sqrt{2\pi}} \, e^{-\frac{1}{2}Z^2} \quad -\infty < z < +\infty .$$

Any normal distribution can be converted into the standard form by the substitution

$$(3) \qquad Z = \frac{x - \mu_X}{\sigma_X} .$$

It is customary to create generators that determine numbers distributed according to the function F(z) and to derive the variable x that has mean μ_X and standard deviation σ_X by the transformation $X = z\sigma_X + \mu_X$. Neither the cumulative distribution function nor its inverse can be expressed in terms of simple mathematical functions but approximation methods are available for their evaluation.

In this report we shall examine the method for generating normal variates which are based on the central limit theorem, (CLT).

6.4 USE OF THE CLT FOR GENERATING NORMAL DEVIATES

In order to simulate a normal distribution with a given expected value μ_X and standard deviation σ_X the following mathematical interpretation of the CLT may be given.

If $r_1, \ldots\ldots, r_N$ are independent random variables each having the same probability distribution with $E(r_i) = \theta$ and $Var(r_i) = \sigma^2$ then

$$(1)\quad \lim_{N\to\infty} P_r\left[a < \frac{\sum_{i=1}^{N} r_i - N\theta}{\sigma\sqrt{N}} < b \right] = \frac{1}{\sqrt{2n}} \int_a^b e - \tfrac{1}{2}z^2 \, dz$$

where

$$(2)\quad E\left(\sum_{i=1}^{N} r_i\right) = N\theta$$

$$(3)\quad \text{var}\left(\sum_{i=1}^{N} r_i\right) = N\sigma^2$$

$$(4)\quad z = \frac{\Sigma r_i - N\theta}{\sigma\sqrt{N}}$$

It follows from the definition of the standard normal distribution (page 119) and from equation 3 (page 119) that z is a standard normal deviate.

The procedure for simulating normal deviates on a computer involves taking the sum of k uniformly distributed random variates $r_1, r_2, \ldots, r_k$ where $0 \leq r_i \leq 1$. The expected value (mean) if the uniform sequence $r_1, \ldots, r_N$ is $\theta = \frac{a+b}{2} = \frac{1}{2}$ and $\sigma = \frac{b-a}{\sqrt{12}} = \frac{1}{\sqrt{12}}$ so (4) becomes

$$(5)\quad z = \frac{\sum_{i=1}^{K} r_i - \frac{K}{2}}{\sqrt{K/12}}$$

But by definition, z is a standard normal variate and can be written

$$(6)\quad z = \frac{X-\mu_X}{\sigma_X}$$

From (5) and (6) we finally have,

$$(7)\quad X = \sigma_X \left(\frac{12}{K}\right)^{1/2} \left(\sum_{i=1}^{K} r_i - \frac{K}{2}\right) + \mu_X$$

Equation (7) now provides us with a simple formula for generating normally distributed random variates with mean equal to μ_X and variance equal to σ_X^2. The value of K is usually taken to be 12 [2] and formula (7) becomes,

$$(8)\quad X = \sigma_X \left(\sum_{i=1}^{12} r_i - 6\right) + \mu_X$$

The above method has been programmed in FORTRAN and subroutine RANDNR produces random normal variates with given expected value and standard

deviation.

Subroutine RANDNR

Purpose:

This subroutine generates a random sequence of normal variates with given mean and standard deviation.

Entry:

RANDNR has two entries:

1) The first is for normal usage.
2) Since RANDNR subroutine calls subroutine PINAX (see program pageC38) to initialize the table from which the function IRAND will pick up the random numbers and additional entry has been provided in order to avoid reinitialization of the subroutine PINAX for each subsequent reference.

Standard Entry:

CALL RANDNR(X,N,EX,STDX,MS)

where	Description	Type
X	is an array of N random numbers normally distributed.	Real (OUTPUT)
N	is the size of the array X	Integer (INPUT)
EX	is the desired mean of the output random numbers.	Real (INPUT)
STDX	is the desired standard deviation	Real (INPUT)
MS	is the starting number of the generator IRAND which is incorporated in subroutine RANDNR.	Integer (INPUT, OUTPUT)

Restrictions

None.

Special Considerations:

In order to generate a sequence of N random numbers normally distributed

with mean μ and standard deviation σ, the following logical sequence of instructions may be used.

```
      DIMENSION X(N)
        MS = 1
        EX = μ
        STDX = σ
        N = K
      CALL RANDNR(S,N,EX,STDX,MS)
```

Secondary Entry

```
      CALL FASTNR(X,N,EX,STDX,MS)
```

The arguments of this entry have the same meaning as above:

We give an example to indicate how the entry FASTNR may be used in a certain program.

```
      DIMENSION X(N)
        MS = L
        EX = μ
        STDX = σ
        NI = K
      CALL RANDNR (X,NI,EX,STDX,MS)
        EX = MI
        STDX = σI
        N2 = KI
      CALL FASTNR (X,N2,EX,STDX,MS)
        END.
```

Other Subprograms Required

IRAND function.

Error Returns

None

Comments

For a flow-chart and coding of sub. RANDNR see pages B9, C37.

6.5 THE EXPONENTIAL DISTRIBUTION

Throughout our daily life we observe time intervals between the occurrences of distinct random events. We receive information about numerous events that take place in our environment such as births, deaths, accidents etc. on the basis of a completely independent time schedule.

If the probability that an event will occur in a small time interval is very small, and if the occurrence of this event is statistically independent of the occurrence of other events, then the time interval between the occurrence of events of this type is exponentially distributed.

A random variable X is said to have an exponential distribution if its density function is defined as,

(1) $$f(x) = ae^{-ax}$$

for $a > 0$ and $x \geq 0$.

The cumulative distribution function of X is

(2) $$F(x) = \int_0^x ae^{-at}\,dt = 1-e^{-ax} \quad ,$$

and the mean and variance of X are given by the following formulas,

(3) $$EX^* = \frac{1}{a} \quad \text{and (4)} \quad VX = \frac{1}{a^2}$$

Since F(x) exists in explicit form, the inverse transformation technique provides a straightforward method.

From (2) we get $r = 1-e^{-ax}$. Because of the symmetry of the uniform distribution r and 1 - r are interchangeable so the above relation becomes $r = e^{-ax}$ (5). Solving for x we get $x = -\frac{1}{a}\log r =$

$= -EX\log r$ (6).

Formula (6) provides us with a simple way of generating exponential variates.

Subroutine EXPR

*The reason for using this notation in preference to the usual notation (i.e. E(x) and V(x) is because of programming considerations.

Purpose:

This subroutine generates exponential variates using the method described above.

Entry

EXPR has two entry points.

1) The first is for normal usage.

2) For the second entry point the same applies as in RANDNR subroutine (see page 121).

Standard Entry

CALL EXPR(X,EX,N,MS).

where	Description	Type
X	is an array of N random numbers exponentially distributed.	Real (OUTPUT)
EX	is the desired mean of the pseudo-random numbers.	Real (INPUT)
N	is the size of the array X	Integer (INPUT)
MS	The same as in RANDNR	Integer (INPUT,OUTPUT)

Restrictions

None

Secondary Entry

CALL EXPQ (X,EX,N,MS)

The arguments of this subroutine have been explained above.

Other Subprograms Required

IRAND function.

Error Returns

None

Comments

For a flow-chart and a program of sub. EXPR see pages B10, C22.

6.6 THE GAMMA DISTRIBUTION

If a process consists of K successive events and if the total elapsed time of this process can be regarded as the sum of K independent exponential variates each with parameter a, the probability distribution of this sum will be a gamma distribution with parameters a and K.

The sum of K (where K is a positive integer) exponential variates each having the same parameter a is also called an Erlang distribution.

Mathematically, the Erlang distribution is a convolution of K exponential distributions. Furthermore, the sum of K exponential variates with identical a is also a gamma distribution if K follows the negative binomial or geometric distribution. The most general form of the gamma distribution arises when K is positive but is not restricted to integral values. The gamma distribution is described by the following density function,

(1) $$f(x) = \frac{a^K x^{K-1} e^{-ax}}{(k-1)!}$$, where $a > 0, K > 0$ and x non-negative.

The mean and the variance of (1) are given by the formulas,

(2) $$EX = \frac{K}{a} \quad \text{and} \quad VX = \frac{K}{a^2} \quad (3)$$

Since the cumulative distribution function for a gamma distribution cannot be formulated explicitely, we must consider an alternative method of generating gamma variates.

Erlang variates may be generated by simply reproducing the random process on which the Erlang distribution is based. This can be done by taking the sum of K exponential variates $x_1, x_2, \ldots, x_K$ with identical expected value 1/a.

Therefore the Erlang variate X can be expressed as

$$(4) \qquad X = \sum_{i=1}^{K} x_i \quad .$$

Since $x_i = -\frac{1}{a} \log r_i$ (see page 123) the above formula becomes,

$$(5) \qquad x = -\frac{1}{a} \sum_{i=1}^{K} \log r_i = -\frac{1}{a}(\log \prod_{i=1}^{K} r_i) \quad ,$$

where $\prod_{i=1}^{K} r_i = r_1 r_2 \ldots . r_K$.

The problem of generating gamma variates when $\underline{K}$ is not an integer will be based on an approximation method.

For example if K = 4.8 we generate a mixture of Erlang variates with K = 4 and K = 5, but with expected value of K = 4.8.

Say, we want to generate N gamma variates and K = A.B where $1 \leq B \leq 9$ (decimal part of A).

We want the expression $Ap_1 + (A + 1)p_2 = A.B$ (1) to be satisfied for p_1, p_2 where p_1 and p_2 are the probabilities to pick up A and A + 1 in such a way so that the mean of A and A + 1 be equal to A.B.

We clearly have the two relations,

$$(6) \qquad \frac{Ax + (A + 1)y}{x + y} = A.B$$

$$(7) \qquad x + y = N$$

Solving this system we get X = N - 0.BN, y = 0.BN

so $p_1 = \frac{N - 0.BN}{N} = 1 - 0.B$ and $p_2 = \frac{0.BN}{N} = 0.B$

In the above example we therefore have $p_1 = 0.2$ and $p_2 = 0.8$, so we must choose K = 4 with probability 0.2 and K = 5 with probability 0.8.

Subroutine ERLANG

Purpose:

This subroutine generates a random sequence of numbers having the Erlang distribution.

Entry:

ERLANG has two entries:

1) The first is for normal usage.

2) For the second entry point the same applies as in RANDNR subroutine (see page 121).

Standard Entry

CALL ERLANG (K,A,X,N,MS)

where	Description	Type
K	is a parameter of the distribution.	Integer (INPUT)
A	This variable is also a parameter of the Erlang distribution.	Real (INPUT)
X	is an array of N numbers having the Erlang-distribution.	Real (OUTPUT)
N	is the size of the array X.	Integer (INPUT)
MS	The same as in RANDNR	Ineteger (INPUT,OUTPUT)

Restrictions

None

Secondary Entry:

CALL ERLANF (K,A,X,N,MS)

The arguments of the entry point ERLANF have the same meaning as above.

Special Considerations

The same as in RANDNR.

Error Returns

None

Other Subprograms Required

IRAND generator

Comments

For a flow-chart and coding of this subroutine see pages B11, C68.

Subroutine GAMMA

Purpose:

This subroutine generates a random sequence of numbers having the GAMMA distribution.

Entry

GAMMA has two entries:

1) The first is for normal usage.

2) For the second entry the same applies as in RANDNR (see page 121).

Standard Entry

CALL GAMMA (RK,A,Y,N,MS)

where	Description	Type
RK	is a parameter of gamma distribution. The meaning of this number has been explained in the text.	Real (INPUT)
A	is a parameter of gamma distribution.	Real (INPUT)
Y	is an array of N random numbers having the gamma distribution.	Real (OUTPUT)
N	is the size of the array.	Integer (INPUT)
MS	the same as in RANDNR	Integer (INPUT)

Restrictions

None

Secondary Entry

CALL GAMMAF (RK,A,Y,N,MS)

The arguments of the entry point GAMMAF have the same meaning as above.

Special Considerations

As in RANDNR.

Error Returns

None

Other Subprograms Required

1) IRAND generator.

2) Subroutine ERLANG. This subroutine produces a mixture of Erlang variates with $K = \lfloor RK \rfloor$ and $K_1 = \lfloor RK \rfloor + 1$ as we have seen in the text.

Comments

As we have seen in the text we choose $K = \lfloor RK \rfloor$ with probability p_1 and $K_1 = \lfloor RK \rfloor + 1$ with probability p_2. To do so in the program, we generate a random number V uniformly distributed between 0 and 1 and compare it with the probability p_1. If $0 \leq V < p_1$ then the ERLANG subroutine - through its entry point ERLANF - will generate the output number Y(I) with parameter $K = \lfloor RK \rfloor$ but if $V \geq p_1$ ERLANF will generate the output number Y(I) with parameter $KI = \lfloor RK \rfloor + 1$.
For a flow-chart and coding of this subroutine see pages B12, C65.

6.7 LOGNORMAL DISTRIBUTION

If the logarithm of a random variable has a normal distribution, the random variable has a continuous distribution known as the lognormal distribution.

If the logarithm (to the base e) of a random variable X has a density function f(y) given as follows:

(1) $f(y) = \frac{1}{\sigma_y \sqrt{2\pi}} \exp\left[\left(-\frac{1}{2}\right)\left(\frac{y - \mu_y}{\sigma_y}\right)^2\right] \quad -\infty < y < +\infty$

where $\log X = y$ and only positive values of X are considered, then X is said to have a lognormal distribution. The parameters μ_y and σ_y correspond to the mean and variance of y, where y is normally distributed variable. The mean and variance of the lognormal distributed variate X are given by the following formulas:

(2) $EX = \exp\left(\mu_y + \frac{\sigma_y^2}{2}\right)$

(3) $VX = (EX)^2[\exp(\sigma_y^2) - 1]$

Simulation of lognormal variates with given mean and variance requires expressing μ_y and σ_y^2 in terms of EX and VX. From equations (2) and (3) can be easily found the relation:

(4) $\sigma_y^2 = \log\left[\frac{VX}{(EX^2)} + 1\right]$ and (5) $\mu_y = \log(EX) - \frac{1}{2}\log\left[\frac{VX}{(EX)^2} + 1\right]$.

Now that μ_y and σ_y^2 have been expressed in terms of the mean and variance of x, the lognormal variate to be generated, the standard normal variate z, can be defined as

(6) $z = \frac{y - \mu_y}{\sigma_y} = \frac{\log x - \mu_y}{\sigma_y}$

From (6) we have $x = \exp(\mu_y + \sigma_y z)$

From the central limit theorem (page 119) we have the equation for $z = \frac{\Sigma r_i - K/2}{\sqrt{K/12}}$ where r_i are uniformly distributed variates between 0 and 1.

Substituting the value of z in (7) and taking K to be equal to 12, formula (7) becomes,

(8) $X = \exp\left[\mu_y + \sigma_y \left(\sum_{i=1}^{12} r_i - 6\right)\right]$.

In order to generate lognormal variates $x_1, x_2, \ldots$ with EX and VX given, we must find μ_y and σ_y from equation (4) and (5) and substitute these values into equation (8).

Subroutine LOGNOR

Purpose:

Subroutine LOGNOR generates lognormally distributed random numbers with given mean and standard deviation.

Entry:

LOGNOR has two entries:

1) The first is for normal usage.

2) For the second entry, the same applies as for RANDNR (see page 121)

Standard Entry

CALL LOGNOR (X,N,EX,STDX,MS)

where	Description	Type
X	is an array of N random numbers lognormally distributed	Real (OUTPUT)
N	is the size of the array X	Integer (INPUT)
EX	is the desired mean of the lognormally distributed numbers	Real (INPUT)
STDX	is the desired standard deviation of the lognormally distributed numbers.	Real (INPUT)
MS	the same as in RANDNR.	Integer (INPUT,OUTPUT)

Restrictions

None

Error Returns

None

Secondary Entry:

CALL LOGN RF (X,N,EX,STDX).

The arguments of the entry LOGNRF have the same meaning as above.

Other Subprograms Required

IRAND generator.

Nomenclature

EY is the mean of the numbers y = logX which are normally distributed.

Vy is the standard deviation of the numbers y = logX.

Comments

For a flow-chart and coding of LOGNOR see pages B13, C66.

6.8 DISCRETE PROBABILITIES DISTRIBUTIONS

A significant number of probability distributions are defined on random variables that take only discrete, non-negative integer values.

The cumulative probability distribution for a discrete random variable X is defined in a manner similar to continuous probability distributions:

$$F(x) = p(X \leq x) = \sum_{X=0}^{x} f(x)$$

where f(x) is the frequency or probability function of x defined for integer x values such that

$$f(x) = p(X=x) \qquad \text{for } x = 0,1,2,....$$

We examine the more important discrete probability distributions.

6.9 THE GEOMETRIC DISTRIBUTION

The geometric distribution has probability function $f(x) = pq^x$, $x = 0,1,2,....$ where p denotes the probability of success and q the probability of failure.

The cumulative distribution is $F(x) = \sum_{X=0}^{x} pq^x$, $X = 0,1,....,x$ (1).

Since $F(x) = p(X \leq x)$ and $p(X = 0) = F(0) = p$ (from (1)) the range of $F(x)$ is therefore $p \leq F(x) \leq 1$.

On the other hand $p(X > x) = 1 - F(x)$ (2) and $p(X > 0) = 1 - F(0) = 1 - p = q$.

Similarly $p(X = 1) = F(1) = p + qp$ and $p(X > 1) = 1 - F(1) = q^2$,

and generally $p(X > x) = q^{x+1}$ and from (2) we have $1 - F(x) = q^{x+1}$ (3).

From (3) $\frac{1-F(x)}{q} = q^x$ and since $0 \leq q^x \leq 1$ the range of $\frac{1-F(x)}{q}$ is unity and we can write $r = q^x$ (4) where r is a uniform number in the range (0,1).

From (4) we have $x = \frac{\log r}{\log q}$ since x must be an integer we truncate the above result, that is we take $x = \lfloor \frac{\log r}{\log q} \rfloor$.

Note: As we have defined the G.D. we consider that x includes the number of failures occuring before 1st success. However x can also be defined to include the no. of failures plus the first success.

In this case we have,

$F(x) = pq^{x-1}$, $x = 1,2.....$ while x is given now by the formula

$x = \lceil \frac{\log r}{\log q} \rceil$

To check this formula we observe that $\lceil \frac{\log r}{\log q} \rceil = x$ iff $x-1 < \frac{\log r}{\log q} \leq x$, that is, $(1-p)^{x-1} > r \geq (1-p)^x$ and since r is uniformly distributed we have $P_r((1-p)^x \leq r < (1-p)^{x-1}) =$

$= \int_{(1-p)^x}^{(1-p)^{x-1}} dy = p(1-p)^{x-1}$ as required.

If the probability of success p is big enough and we need more accuracy we can use the following method which is based on the rejection method (page 117).

First, we define a variable x, which is to be used as a counter and set it equal to 1. (In this case we suppose that x contains the number of failures plus the 1st success) Then we generate a sequence of uniform variates $r_1, r_2, \ldots, r_i, \ldots$ terminating the sequence when we

reach a value of r_i less than p. For each value of r_i greater than p, we increase x by one. That is we count the number of failures or r_i's greater or equal to p. When we reach the first value of r_i less than p, the sequence is terminated, and the value of x corresponds to the value of the geometric variate. Two subroutines are constructed. The first generates geometric variates using the formula $x = \lceil \frac{\log r}{\log q} \rceil$ while the second subroutine generates geometric variates when p is large .

Subroutine PASCAL

Purpose:

This subroutine generates geometric variates using the formula

$$x = \lceil \frac{\log r}{\log q} \rceil$$

Entry

PASCAL has two entries:

1) The first is for normal usage.

2) For the second entry the same comments apply as in RANDNR (see page 122).

Standard Entry

CALL PASCAL(Q,IX,N,MS)

Where	Description	Type
Q	is the probability of failure	Real (INPUT)
IX	is an array of N random numbers having the geometric distribution.	Integer (OUTPUT)
N	is the size of the array IX	Integer (INPUT)
MS	is the starting number of the generator IRAND which is incorporated in to the sub. PASCAL.	Integer (INPUT,OUTPUT)

Secondary Entry:

CALL PASCQ (Q,IX,N,MS)

The arguments of the entry point PASCQ have the same meaning as above.

Error Returns

None

Other Subprograms Required

IRAND generator.

Comments

For a flow-chart and coding of the subroutine PASCAL see pages B14, C67.

Subroutine PASBIG

Purpose:

This subroutine generates geometric variates using the rejection method (page 118). This method is usually preferred when the probability of success p is very large.

Entry

PASBIG has two entries:

1) The first is for normal usage.

2) For the second entry the same applies as RANDNR (page 125).

Standard Entry:

CALL PASBIG (P,IX,N,MS)

Where	Description	Type
P	is the probability of success.	Real (INPUT)
IX	is the array of N random numbers geometrically distributed.	Integer (OUTPUT)
N	is the size of the array IX	Integer (INPUT)
MS	the user must set MS equal to any number in the range $[0,2^{35})$. See RANDNR (Page 121).	Integer (INPUT,OUTPUT)

Secondary Entry:

CALL PAFAST (P,IX,N,MS)

The arguments of the entry point PAFAST have been explained above.

Restrictions

None

Error Returns

None

Other Subprograms Required

IRAND generator.

Comments

For a flow-chart and coding of the subroutine PASBIG see pages B17, C74.

6.9 NEGATIVE BINOMIAL DISTRIBUTION

Suppose we have a series of Bernoulli trials repeated until K successes occur (K > 1) then the random variable denoting the number of failures will have the negative binomial distribution. Negative binomial variates are therefore the sum of K geometric variates. In this case K is an integer and the distribution is called a Pascal distribution.

The probability function for the negative binomial distribution is

$$f(x) = \binom{K+x-1}{x} p^K q^x \qquad x = 0,1,2,.....$$

where K is the number of successes out of K+x trials and x is the number of failures that occur before K successes occur.

The expected value (mean) and variance of X are given by

(1) $$EX = \frac{K}{p}$$

(2) $$VX = \frac{K}{p^2}$$

From these two relations we find

(3) $$p = \frac{EX}{VX}$$

(4) $$K = \frac{EX^2}{VX-EX}$$

Now if the computed value of K in equation (4) is not an integer the simulation procedure is complicated considerably.

When K is an integer, Pascal variates can be generated by taking the sum of K geometric variates.

We have $x = \dfrac{\sum_{i=1}^{K} \log r_i}{\log q} = \dfrac{\log\left(\prod_{i=1}^{K} r_i\right)}{\log q}$

If K is not an integer, we must rely on approximation methods for generating negative binomial variates. One method, which we shall use to generate negative binomial variates is exactly the same as we used when we were generating gamma variates (see page 126. We will generate a mixture of variates with two different integral values of K.

For example if K = 3.4 we generate a mixture of Pascal variates with K = 3 and K = 4 but with expected value of K = 3.4.

So we choose K = 3 with probability 0.6 and K = 4 with probability 0.4.

The following two subroutines put the above theory in action and generate negative binomial variates with K = integer and K = real respectively.

Subroutine NBIN

Purpose:

This subroutine generates a random sequence of negative binomial variates with K = integer.

Entry

NBIN has two entries:

1) The first is for normal usage.

2) For the second the same applies as in RANDNR (page 121).

Standard Entry

```
CALL NBIN (K,Q,IX,N,MS)
```

Where	Description	Type
K	is the parameter K of the negative binomial distribution. It represents the number of successes.	Integer (INPUT)
Q	is the probability of failure	Real (INPUT)
IX	is the output array of N random numbers binomially distributed.	Integer (OUTPUT)
N	is the size of the output array	Integer (INPUT)
MS	the user must set MS equal to any number in the range $[0,2^{35})$ see RANDNR.	Integer (INPUT,OUTPUT)

Restrictions

None

Error Returns

None

Secondary Entry

CALL NBINF (K,Q,IX,N,MS)

The arguments of the entry NBINF have the same meaning as above.

Other Subprograms Required

IRAND generator.

Comments

For a flow-chart and coding of NBIN see pages B15, C67.

Subroutine RBIN

Purpose:

This subroutine generates a random sequence of numbers following the negative binomial distribution when the parameter K ≠ integer.

Entry

RBIN has two entries:

1) The first is for normal usage.

2) For the second see RANDNR (page 121).

Standard Entry

CALL RBIN (RK,Q,IY,N,MS)

Where	Description	Type
RK	is the parameter of the negative binomial distribution.	Real (INPUT)
Q	is the probability of failure	Real (INPUT)
IY	is an array of N numbers having the negative binomial distribution.	Integer (OUTPUT)
N	is the size of the array IY	Integer (INPUT)
MS	the user must set MS equal to any number in the range $[0,2^{35})$. see RANDNR (page 121).	Integer (INPUT,OUTPUT)

Restrictions

None

Secondary Entry:

CALL RBINF (RK,Q,IY,N,MS)

The arguments of the entry RBINF have the same meaning as above.

Error Returns

None

Other Subprograms Required

1) IRAND generator.

2) Subroutine NBIN. This subroutine generates a mixture of negative binomial variates with $K = \lfloor RK \rfloor$ and $K_1 = \lfloor RK \rfloor + 1$ as explained in the text.

Comments

As we have seen in the text (page 137) we choose $K = \lfloor RK \rfloor$ with probability p_1 and $K_1 = \lfloor RK \rfloor + 1$ with probability p_2. To do so in the program, we generate a random number U uniformally distributed between 0 and 1 and compare it with the probability p_1. If $0 \leq U \leq p_1$ the NBIN subroutine - through its entry point NFINF - will generate the output number IY with parameter $K = \lfloor RK \rfloor$ but if $U \geq p_1$

the entry NBINF will generate the output number IY with parameter $K_1 = \lfloor RK \rfloor + 1$. For a flow-chart and coding of the sub. RBIN see pages B16, C74.

6.10 THE HYPERGEOMETRIC DISTRIBUTION

Consider a population consisting of N elements such that each element belongs either to class I or class II. Let Np denotes the number of elements belonging to class II, where $p + q = 1$. If a random sample of $n < N$ elements is taken from the population of N elements without replacement, then x, the number of class I elements in the sample of n elements, has the hypergeometric distribution [2]. Applications of the hypergeometric distribution are found in the areas of quality control and production control. The hypergeometric distribution is described by the following probability function.

$$(1) \qquad f(x) = \frac{\binom{Np}{x}\binom{Nq}{n-x}}{\binom{N}{n}} \qquad \begin{matrix} 0 \leq x \leq Np \\ 0 \leq n-x \leq Nq \end{matrix}$$

where x, n, N integers.

The generation of hypergeometric variates involves simulating a sampling experiment without replacement. That is, we merely alter the Bernouilli trials method of generating binomial variates, so that N and p vary depending respectively on the total number of elements that have been previously drawn from the population and the number of class I elements that have been drawn.

As each element in a sample of n elements is drawn, the original value of $N = N_o$ is reduced according to the formula,

$$N_i = N_{i-1} - 1 \quad i = 1,2,\ldots,n.$$

In a similar manner, the value $p = p_0$ when the i^{th} element in a sample of n elements is drawn becomes,

$$(2) \qquad p_i = \frac{N_{i-1}\, p_{i-1} - S}{N_{i-1} - 1} \qquad i = 1,2,\ldots\ldots,n$$

where S = 1 when sample element (i-1) belongs to class I and S = 0 when sample element (i-1) belongs to class II.

The starting values N_0 and p_0, of course, correspond to N, the initial population size, and p, the proportion of the total population consisting of class I elements.

Subroutine HYPGEO

Purpose:

This subroutine generates a random sequence of integers following the hypergeometric distribution.

Entry:

HPYGEO has two entries:

1) The first is for normal usage.

2) For the second the same applies as in RANDNR (see page 121).

Standard Entry:

CALL HYPGEO (NP,M,P,IX,N,MS)

Where	Description	Type
NP	is the population size	Integer (INPUT)
M	is the sample size	Integer (INPUT)
P	is the proportion of the total population consisting of CLASS-I elements.	Real (INPUT)
IX	is an array of N random numbers following the hypergeometric distribution.	Integer (OUTPUT)
N	is the size of the array IX	Integer (INPUT)
MS	the user must set MS equal to any number in the range $[0,2^{35})$. See RANDNR (page 121).	Integer (INPUT,OUTPUT)

Secondary Entry:

CALL HYPGEF (NP,M,P,IX,N,MS)

The arguments of HYPGEF have the same meaning as above.

Restrictions

None

Error Returns

None

Comments

For a flow-chart and coding of the sub. HYPGEO see pages B18, C68.

6.11 POISSON DISTRIBUTION

This distribution is related to the exponential distribution as the binomial distribution is related to the geometric. It represents the number of occurrences, per unit time of an event which can occur at any instant of time; for example, the number of alpha particles emitted by a radioactive substance in a single second has a Poisson distribution. If we take a series of n independent Bernoulli trials, in each of which there is a small probability p of an event to occur, then as $n \to \infty$ (n represents the trials) the probability of x occurrence, is given by the formula

$$(1) \qquad f(x) = e^{-\lambda} \frac{\lambda^x}{x!} \qquad x = 0,1,2,.....$$

when we allow p to approach zero in such a manner that $\lambda = np$ remain fixed.

Poisson distribution has (1) as a probability function. The mean and the variance of the Poisson distribution are equal to parameter λ. To simulate a Poisson distribution with parameter λ, we take advantage of the relationship between the exponential and Poisson distribution. We describe this relationship.

It is known [2] that the probability of x events occuring during

time t is given by the formula $F(x) = e^{-\lambda t} \frac{(\lambda t)^x}{x!}$ (2)

(Another form of Poisson distribution).

If we put x = 0 in (2) then the probability of no events occurring in the time interval t will be $F(0) = e^{-\lambda t}$.

If we now symbolize with T the waiting time of an event to occur then $p(T>t) = e^{-\lambda t}$ and $p(T\leq t) = 1 - e^{-\lambda t}$ and $\frac{dp(T\leq t)}{dt} = \lambda e^{-\lambda t}$ which is the exponential distribution.

Consider now a time horizon that has been divided into unit time intervals as illustrated in Figure 6.1.

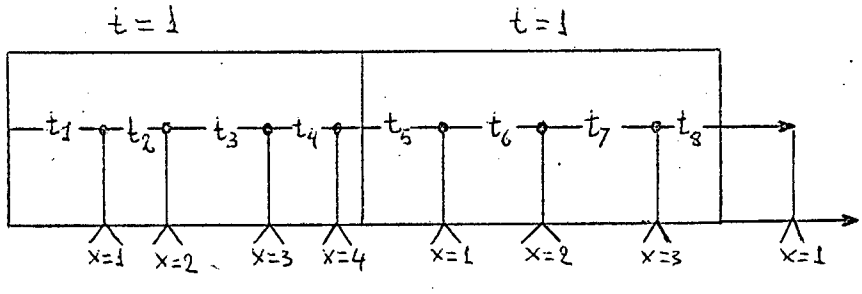


Figure 6.1. Poisson distributed events on a time scale.

Events are assumed along the time horizon and are denoted by a symbol (^). The time interval t between events is assumed to have an exponential distribution with mean equal to $1/\lambda$. This implies that the number of events x occuring during a unit time interval follows a Poisson distribution with mean equal to λ. So we generate exponentially distributed time intervals $t_1, t_2, \ldots$ with expected value to equal to 1. These random time intervals are accumulated as they are generated until their sum exceeds λ (mean of Poisson distribution).

This can be described mathematically as follows:

$$(3)\quad \sum_{i=0}^{x} t_i \leqslant \lambda < \sum_{i=0}^{x+1} t_i \qquad x = 0,1,2,\ldots\ldots \text{ (x = Poisson variate)}$$

The exponential variates t_i are generated by the formula $t_i = \log r_i$ where r_i are uniformly distributed between 0 and 1 (see page 123) with unit mean.

From (3) we have $\prod_{i=0}^{x} r_i \geqslant e^{-\lambda} > \prod_{i=0}^{x+1} r_i$ (4).

So the following algorithm is fully justified in order to generate Poisson variates.

Algorithm: Poisson

Step 1

Set $p = e^{-\lambda}$, $x = 0$, $q = 1$

Step 2

Generate a random variable U, uniformly distributed between zero and one.

Step 3

Set $q = qU$

Step 4

If $q \geqslant p$ set $x = x+1$ and return to step 2. Otherwise the algorithm terminates with output x.

Subroutine POISSN

Purpose:

This subroutine generates a sequence of random numbers having the Poisson distribution.

Entry

POISSN has two entries:

1) The first is for normal usage.

2) For the second the same applies as in RANDNR (see page 121).

Standard Entry:

CALL POISSN (P,IX,N,MS)

Where	Description	Type
P	is the desired mean of the Poisson variates.	Real (INPUT)
IX	is an array of numbers following the Poisson distribution.	Integer (OUTPUT)
N	is the size of the array IX.	Integer (INPUT)
MS	is the starting value of the generator IRAND which is incorporated into subroutine.	Integer (INPUT,OUTPUT)

Restrictions

None

Error Returns

None

Secondary Entry:

CALL POISSF (P,IX,N,MS)

The arguments of the entry point POISSF have the same meaning as above.

Other Subprograms Required

IRAND generator.

Comments

For a flow-chart and coding of the sub. POISSN see pages B19, C68.

CHAPTER 7.

7.1 INTRODUCTION.

In the preceding chapter we have been concerned with techniques for generating stochastic processes on a binary computer. In this chapter two of these generators, the EXPR and RANDNR will be tested in two simulation problems with excellent results. Eventually the behaviour of the generator IRAND is examined, since the good performance of the generators of the preceding chapter depends on the randomness of the IRAND generator.

7.2 SINGLE-STATION QUEUING PROBLEM.

The length of a waiting line depends primarily on time; i.e., under fixed conditions of customer arrivals and service facilities, queue length is a function of time.

Hence the process of waiting formation is sometimes referred to as a stochastic process.

Both arrival time and service time are assumed to be stochastic variates (following the exponential distribution) with known parameters.

The variables of our model of a single channel queuing system are defined below.

AT_i = the time interval between the arrival of the ith unit and the (i+1)th unit, i=1,2,..,m.

ST_i = the service time for the ith arrival unit i=1,2,..,m.

WT_i = the amount of time the ith arrival unit spends waiting to enter the service station, i=1,2,..,m.

IDT_i = the amount of time the service station remains idle while waiting for the ith arrival unit to arrive i=1,2,..,m.

TWT_i = total waiting time when the ith arrival unit enters the service station.

$TIDT_i$ = total idle time when the i arrival unit enters the service station.

When the first input unit arrives at the service station - that is when i=1 - the following starting conditions are assumed to be in effect.

TIDT1=0

TWT1 =0

IDT=0

WT=0

AT=0

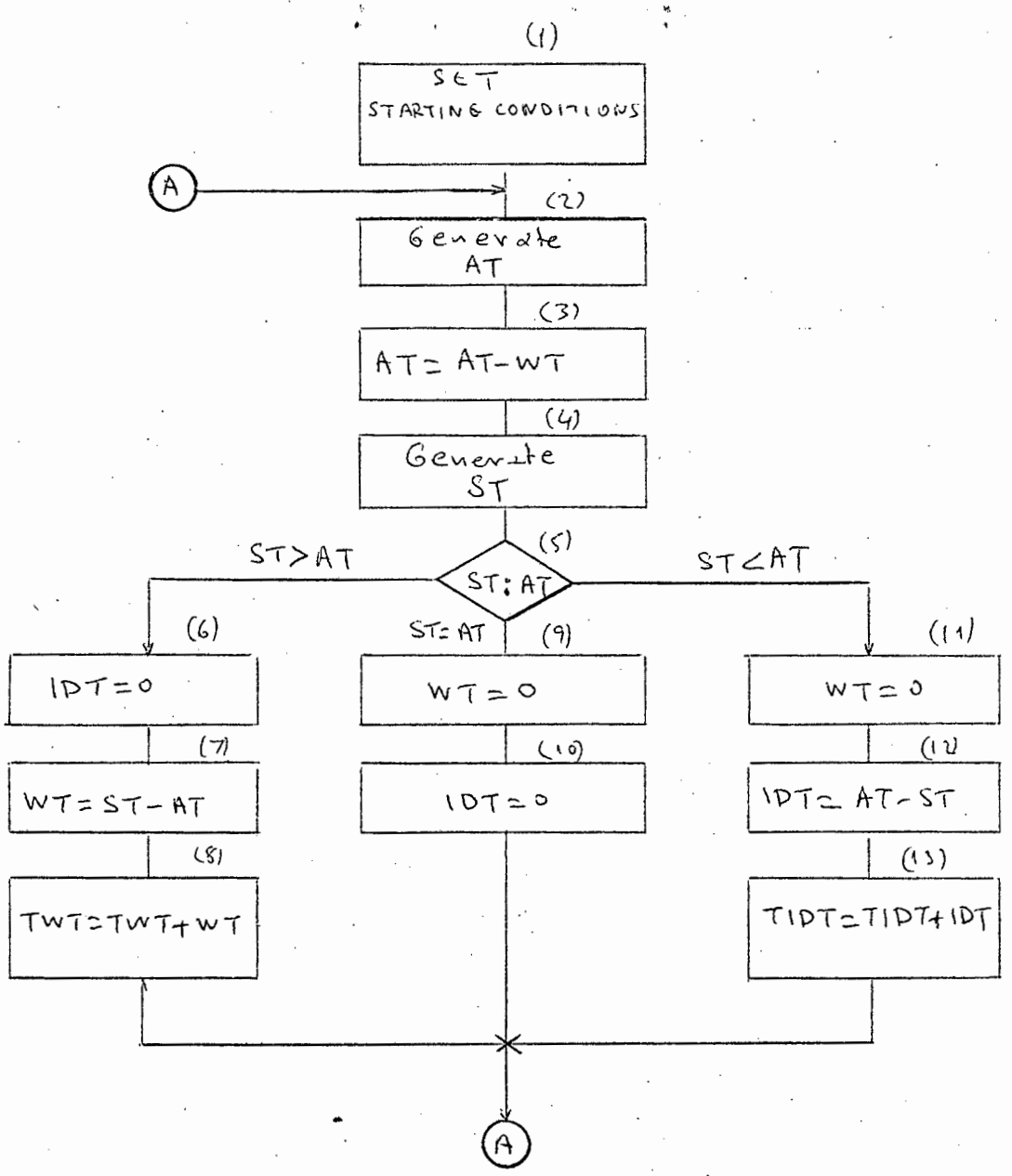
Figure 7.1 contains a computer flowchart of the single-channel queuing system.

In block 1 AT,WT,IDT,TWT,TIDT are set equal to zero, indicating that the first unit has arrived at the service station.

A second unit is assumed to arrive at the system and the arrival time is generated in Box 2.

Waiting time, WT, is subtracted from arrival time in block 3. If service time exceeds arrival time, the second or (i+1)th input unit arrives before service is completed on the first or ith unit. Therefore waiting time occurs and idle time is set equal to zero. Waiting time is set equal to the difference between service and arrival time and accumulated in blocks (7),(8) respectively. On the other hand, if service time is less than arrival time, idle time results, and waiting time is equal to zero. Idle time is then set equal to the difference between arrival and service time and accumulated. If service time and arrival time are equal, neither waiting or idle time occur.

The procedure can be repeated for as many arrivals as are required



Flowchart: Variable time increment model.

or for as long a period of time as necessary.

At the end of the simulation run, statistics such as expected waiting time, expected idle time, etc., can be computed and compared with known theoretical values of these parameters.

For example it can be shown [17] that for the case in which arrival time and service time , both have exponential distributions with expected values equal to $1/\lambda$ and $1/\mu$ respectively that the following relationships hold when $\mu>\lambda$.

(1) $\frac{\lambda}{\mu(\mu-\lambda)}$ = expected waiting time .

(2) $\frac{1}{\mu-\lambda}$ = expected time an arrival spends in the system .

Program MODEL (see pageC69) based on the flowchart on page 149 simulates the situation for 10,000 arrivals. The mean arrival time is taken to be 50 and service time 35. So $\lambda=\frac{1}{50}$ and $\mu=\frac{1}{35}$ and taking into account formulas (1), (2) we find, expected waiting time = 81.6; expected time an arrival spends in the system = 116.6.

Since the mean arrival time is 50 and service time 35, the mean idle is expected to be 15.

The output results of this program are presented on page C70.

We found that,

expected waiting time = 78.88

expected time an arrival = 113.86
spends in the system.

expected idle time = 15.30.

The simulated values are very close to theoretical ones.

In program MODEL we use subroutine EXPR to generate the arrival and service time.

7.3 A MULTICHANNEL PROBLEM.

Consider a system consisting of n service stations operating in parallel. Input units arrive at the system and are admitted to the first vacant service station on a first come first served basis.

The time interval between arrivals is a stochastic variate following the exponential distribution. The service time for each of the n stations also follows the exponential distribution.

When an input system arrives at the system the n service stations are checked to determine whether any one of them is vacant at the moment.

If all n are occupied, then waiting time occurs until one station becomes vacant. When a service station becomes vacant before another unit arrives, idle time occurs until a unit arrives and enters the vacant service station.

The variables of our multichannel model (which are required to be integers) are defined below:

AT, ST and IDT denote the arrival, service and idle time respectively.

(1) TAT_i = total arrival time when the ith arrival unit arrives at the system, i=1,2,...

(2) $T_{ij} = ST_{ij} + IDT_{ij}$
= the time interval between the departure of the (i-1)th unit and the ith unit from the j station, i=1.2.,, j=1,2,..,n

(3) TT_{ij} = total time that has elapsed at the jth service station when the ith arrival unit departs from the jth station, i=1,2,..,n.

(4) SMIN = the minimum $TT_{i-1,j}$ over all j (j=1,2,..,n).

When the first unit arrives at the system, the following relationships hold:

$AT1=0$

$IDT_{1,j}=0 \qquad j=1,2,..,n$

(5) $WT_{1,j}=0 \qquad j=1,2,..,n$

$TT_{11}=ST_{11}$

$ST_{1j}=0 \qquad j=2,..,n$

(WT denotes the waiting time).

For all subsequent arrivals the following relationships describe the system. If

(6) $TAT_i > SMIN \qquad i=1,2,..$

then

(7) $IDT_{ij}=TAT_i-SMIN \qquad i=1,2,..,\ j=1,2,..,n$

and

(8) $WT_{ij}=0$

If on the other hand,

(9) $TAT_i \leq SMIN \qquad i=1,2,...$

then

(10) $WT_{ij}=SMIN-TAT_i$

and

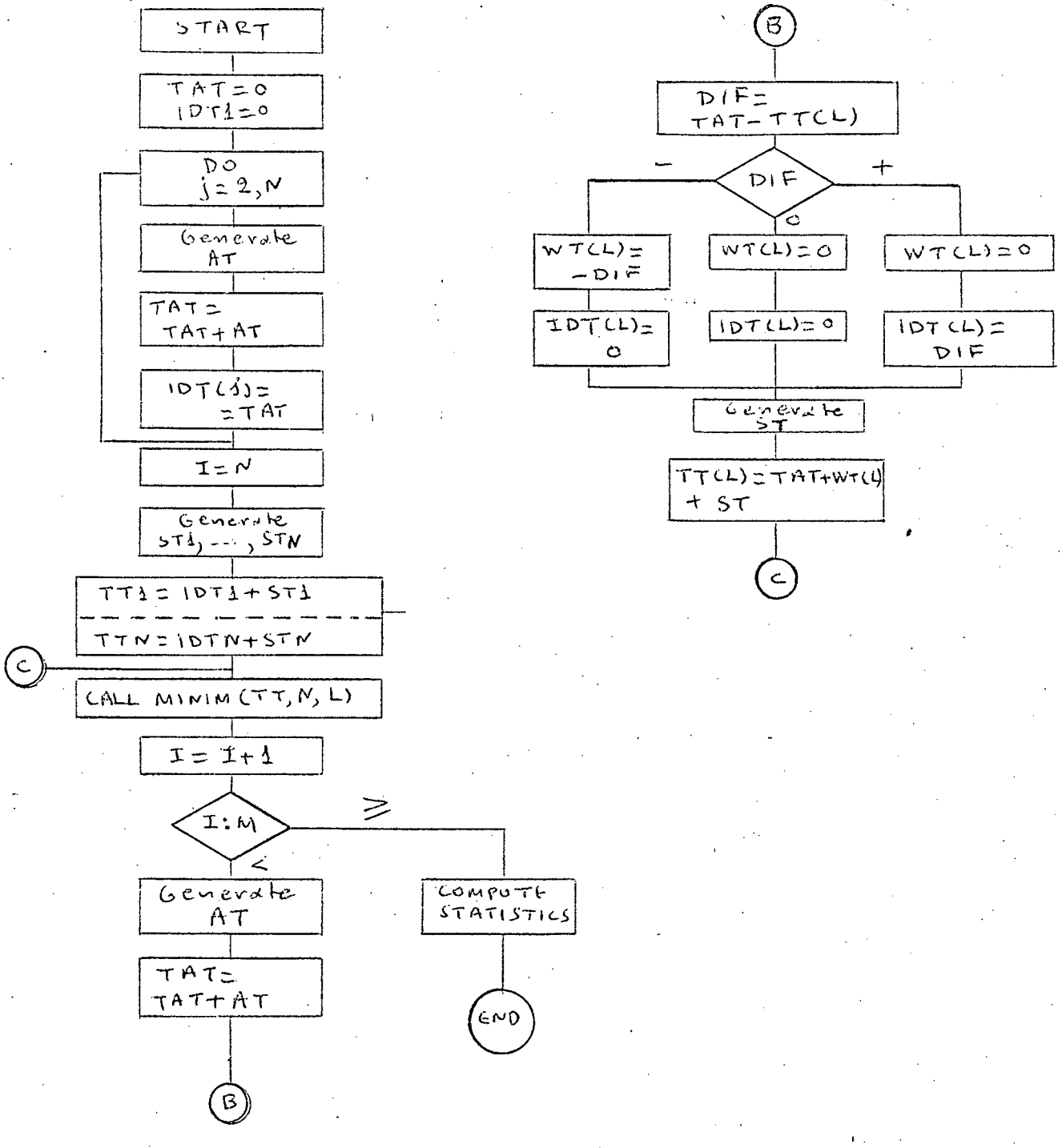
(11) $IDT_{ij}=0$

Figure 7.2 contains a flowchart for simulating a multichannel queuing system.

The variable M denotes the total number of arrivals, while N denotes the total number of service stations.

In Box 10 of figure 7.2 subroutine MINIM (see page C31 for a program) finds the position L of the SMIN in the array TT.

FIG.7.2 A MULTIPLE CHANNEL MODEL.



7.4 AN APPLICATION OF THE MULTICHANNEL MODEL IN THE AIRCRAFT INDUSTRY.

In the aircraft industry usually arises the problem of the optimum number of clerks that should be placed in the company's factory tool cribs.

For one particular tool crib, the average time between arrivals of mechanics was found to be 35 seconds. Also, for this same crib the average serving time for the individual mechanics was found to be 50 seconds.

We use λ to stand for the average arrival rate, and μ to stand for the average service rate over a fixed length of time. If we select the average serving time as this fixed unit, it follows that:

$$\lambda = \frac{\frac{1}{35}}{\frac{1}{50}} = \frac{\text{mechanics arriving per second}}{\text{average serving time per second}} = 1.43 \text{ mechanics}$$

arriving per average serving time, i.e. the average time of arrivals per 50 seconds is 1.43 mechanics.

The average serving rate for 50-second time intervals is of course, $\mu=1$.

To find the average waiting time T_w (in average serving time units of 50 seconds) we use the following equations [15],

$$(1)\quad T_w = \frac{P_o}{s\mu(s!)[1-(\lambda/s\mu)]^2} \cdot \left(\frac{\lambda}{\mu}\right)^s$$

where P_0= the probability that there are no units in the system at a particular time =

$$= \frac{1}{\sum_{n=0}^{s-1} (\lambda/\mu)^n/n! + [(\lambda/\mu)^s/[s!(1-\lambda/\mu s)]]}$$

where s refers to the number of clerks at the counter.

Table I below illustrates the calculations made for two, three and four clerks.

Since on the average one caller arrives every 35 seconds, in a working day of 7.5 hours, the expected number of arrivals is

$$\frac{(7.5)(3600)}{35} = 770.$$

TABLE I

λ	μ	S	P_0	T_w	
				Average serving time units	Seconds
1.43	1	2	0.116	1.04	52.0
1.43	1	3	0.228	0.135	8.8
1.43	1	4	0.237	0.025	1.3

For this number of arrivals there would be required at the rate of 50 seconds service for each a total of

$$\frac{770(50)}{3600} = 10.7 \text{ hours}$$

of service on the part of the clerks in one working day.

So it is apparent that at least two clerks must exist to handle the tool crib.

If there were two clerks furnishing 15 hours of service (working day = 7.5 hours), there would be 15-10.7 or 4.3 hours of idle time on the part of these clerks.

But for two clerks we have seen (see above Table) that the expected waiting time for each mechanic is 52 seconds. Hence for an expected number of 770 arrivals per day, the expected waiting time would be 770x52 sec. or 11.1 hours.

If we let 2 rand per hour represent the labour cost of clerks and 5 rand per hour the cost of a mechanic, then the total cost would be 64.10 rand.

For three clerks the cost can be similarly computed. It turns out to be 31 rand. However, for four clerks, the total cost rises to 40 rand, and the cost continues to rise for additional clerks.

Hence the optimum number of clerks is three.

Apparently this problem can be solved using simulation based on the flowchart of page 153.

Subroutine MULCHA below translates this flowchart into a FORTRAN program.

SUBROUTINE MULCHA.

Purpose:

This subroutine simulates the multichannel queuing model, and given the labour cost of a clerk per hour and the cost per customer (mechanic) this subroutine computes the total cost of the waiting time on the part of clerks and mechanics.

Entry:

MULCHA has one entry:

CALL MULCHA(M,N,MS,EX1,EX2,PRICCL,PRICME,TCOST,TWT,TIDT,COSTC,COSTM).

where	Description	Type
M	is the total number of customers (mechanics).	integer.(INPUT)
N	is the total number of clerks	integer.(INPUT)

MS	The user must set MS equal to a number in the range $[0,2^{35})$. This number is used by the sub. EXPR which is contained into the sub-routine MULCHA.	integer. (INPUT OUTPUT)
EX_1	is the mean arrival time.	Real. (INPUT)
EX_2	is the mean service time.	Real. (INPUT)
PRICCL	is the labour cost of a clerk per hour.	Real. (INPUT)
PRICME	is the labour cost of a mechanic per hour.	Real. (INPUT)
TCOST	is the total labour cost on the part of clerks and mechanics.	Real. (OUTPUT)
TWT	is the total waiting time in hours.	Real (OUTPUT)
TIDT	is the total idle time in hours.	Real (OUTPUT)
COSTC	is the total labour cost on the part of clerks.	Real. (OUTPUT)
COSTM	is the total labour cost on the part of mechanics.	Real. (OUTPUT)

Restrictions.

None.

Error returns.

None

Other subprograms required.

1. Subroutine EXPR (page 123) which generates the arrival and service time.

2. Subroutine MINIM. This is analogous to subroutine MAXIM (see page C30).

   MINIM finds the position of the minimum $TT_{i-1,j}$ over all j, j=1,2,..,n. (page 151 relation 4).

For a coding of subroutine MULCHA see page C71.

Program MULCHAPROG (see page C56) calls subroutine MULCHA which outputs the total cost when the number of clerks N equals to 2. We increase N by 1 and again MULCHA is called to calculate the new total cost which is compared with the previous one. The iteration scheme stops when a total cost is greater than the previous one.

Details of the output of the program MULCHAPROG can be found on page C59.

The simulation results agree with the theoretical ones. In fact when we use 3 clerks, the total cost is 31.943 rand which is less than 53.649 or 38.864 where 2 or 4 clerks were used respectively. Therefore, the optimum number of clerks is three.

7.5 AN APPLICATION OF NORMAL DEVIATES.

Problem:

The length of life of 100 vacuum tubes contained in a digital computer is normally distributed with expected value equal to 180 days and standard deviation equal to 14 days. If all the tubes are replaced at one time the cost of replacing them is $2 per tube.

The cost of replacing individual tubes that fail in service is $5

per tube plus the cost of computer down-time. On the average, the cost per tube for down-time is $50 during the day and $100 at night The probability of a failure during the day is 0.7 and the probability of a failure at night is 0.3. Use computer simulation to compare the cost of the following two policies: (1) Replace the tubes individually as they fail, and (2) replace all 100 tubes every five months and replace the tubes that fail during the interim period individually. Determine an optimum replacement policy for the firm.

Figure 7.3 present a flow-chart which offers an answer to the first question of the above problem.

In Box 1 initialization takes place. The total cost, COST, of replacing individual tubes is set equal to zero. Similarly FDAY and FNIGHT. FDAY is a storage location which will contain at the end of the computation, the number of tubes which failed during the day and FNIGHT the number of tubes failed at night.

In Box 2 we generate N normal deviates W(k) with given mean and standard deviation and assign these variates to each of the N tubes. These variates represent the length of life of N vacuum tubes.

In Box 3 we sort them into ascending order. The decision Box 5 asks if W(k) is less or equal to TIME.
TIME denotes the total simulation time of the experiment.
If yes, a random number υ is generated and it is compared with the probability P (P denotes the probability to have a failure during the day). If $\upsilon<P$, FDAY is increased by one and the cost is calculated, but if $\upsilon\geq P$ then FNIGHT is increased by one and the COST (Box 1) is computed. In either case, we go to Box 12 where a new normal variate is generated and added to W(k). ST represents the new length of life

of a particular tube.

The counter k is increased by one - we examine a new tube - and the action is sent to Box 5 if $k \leq N$ or to Box 16 if $k > N$.

If now the answer is no (Box 5) we sort W(k) into ascending order (Box 16) and we ask (Box 17) if W(1) is less or equal to TIME. If yes, we go to Box 6 and the whole process is repeated again, but if W(1) is greater than TIME, the process terminates.

In the figure 7.3, DOWND represents the cost of computer down-time during the day and DOWNN the cost during the night.
PRICE denotes the cost of an individual tube.

Subroutine below puts in action the flowchart of figure 7.3.

Subroutine Tube

Entry:

This subroutine has one entry:

CALL TUBE(N,TIME,EX,STDX,DOWND,DOWNN,P,PRICE,FDAY,FNIGHT,COST,MS)

EX,STDX are the mean and standard deviation which are required by the subroutine RANDNR which is incorporated into sub. TUBE, RANDNR generates normal variates. (See page 121).

The argument MS is required by the sub. RANDNR.
The user must set MS equal to an arbitrary integer from the range $[0, 2^{35})$.
The meaning of the other arguments have been previously explained.

The output arguments of the sub. TUBE are, FDAY,FNIGHT, and COST.
The arguments FDAY and FNIGHT are considered to be FORTRAN integers.

Program TUBEPROG (see page C73) calculates first the total cost when the tubes are replaced individually as they fail. The total cost in this case is R26,130. In the same program the total cost for the

second case is computed, that is, when the tubes are replaced every 5 months. This cost is found to be RI 070. Apparently the optimum replacement policy for the firm is, to replace all 100 tubes every 5 months and replace the tubes that fail during the interim period individually.

Details of the output of this program can be found on page C70.

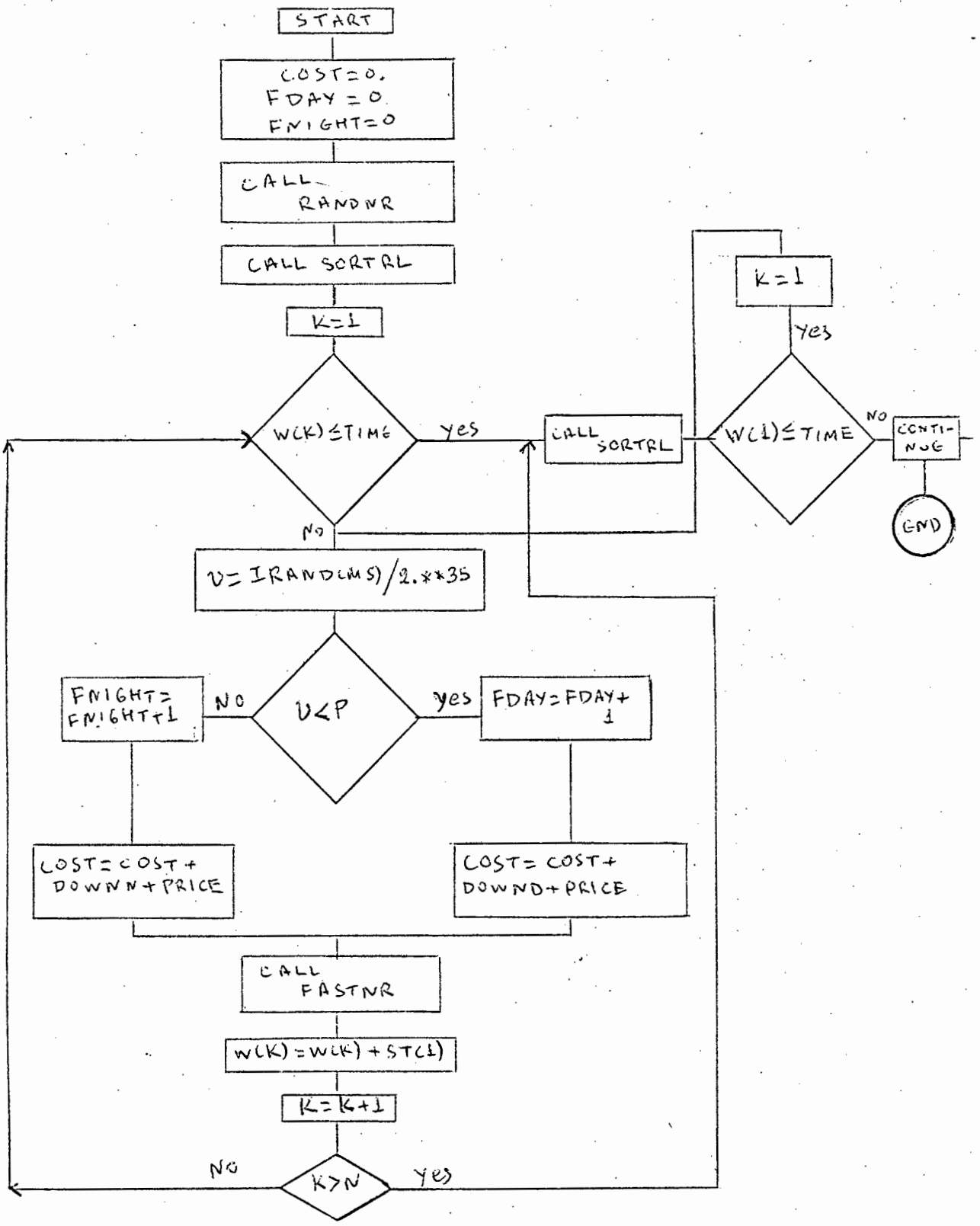


Figure 3. Flowchart of subroutine TUBE.

CHAPTER 8

CONCLUSIONS

IRAND generator was succesfully tested with a wide variety of statistical tests and simulation problems.
In chapter 2 was proved that IRAND is faster than the original Marsaglia-MacLaren generator. In chapter 3 both generators were subjected to a series of thirteen statistical tests. It was shown there that IRAND improves the randomness of the original generator.

Chapters 4 and 5 were devoted in testing IRAND generator in some practical applications.

The produced results were close enough to the theoretical ones.

In chapter 6 generator IRAND was used as a base of generating some of the most known probability distributions, discrete and continuous. Two of these generators, the RANDNR and EXPR - they generate normal and exponential variates respectively - where tested in chapter 7 in one Monte-Carlo problem the first and in two queuing models the second. The results were excellent compared with the theoretical ones.

Even in this chapter the effectiveness of the IRAND generator is proved, since the good performance of the above generators depends on the randomness of the IRAND generator which produces random numbers uniformly distributed in the range [0,1).

Concluding we can say that generator IRAND has a high probability of being satisfactory and can be strongly recommended as a source of random numbers.

BIBLIOGRAPHY

1 Blackwell D. and Girshick, M.A. "Theory of Games and Statistical Decisions". John Wiley and Sons, New York, 1954.

2 Cacoullos T. "Probability Theory". University of Athens, 1970.

3 Coveyou R.R. and Mac-Pherson R.D. "Fourier Analysis of Uniform Random Number Generators". J.A.C.M., Vol. 14, No. 1, January, 1967.

4 Cramer H. "Mathematical Methods of Statistics". Princeton W.J. Prin. Univer. Press, 1947. Page 232.

5 Feller W. "An Introduction to Probability Theory and its Applications".

6 Greenberger M. "Notes on a new Pseudo-Random Number Generator". J.A.C.M., Vol. 8, 1961, pp 163-167.

7 Gordon G. "System Simulation". Prentice Hall Series in Automatic Computation.

8 Hardy and Wright. "The Theory of Numbers". 4th Edition. (Oxford, 1960) Chapter 22.

9 International Business Machines Corporation. "Random Number Generation and Testing". Reference Manual (N. York, 1959).

10 Janssou, Birger. "Random Number Generators". Stockholm: Almguist and Wiksell, 1966.

11 Knuth D.E. "The Art of Computer Programming". Volumes 1 and 2.

12 Marsaglia G., MacLaren M.D. J.A.C.M. (1965) pp. 83-89.

13 Marshall A.W. "Experimentation by Simulation and Monte-Carlo". The RAND Corporation.

14 Naylor, Thomas H., Burdick D., Kong Chu. "Computer Simulation Techniques". New York: John Wiley and Sons. 1966.

15 Saaty T.L. "Resumé of Useful Formulas in Queuing Theory". Operation Research, V (Apr. 1957) pages 162-187.

16 Sceid F. "Numerical Analysis". (SHAUM'S OUTLINE SERIES).

17 Sasieni, Maurice, Yaspin, Arthur and Friedman, Lawrence. "Operation Research". New York, Wiley and Sons, 1959.

18 Tocher K.D. "The Art of Simulation". Princeton N.J. P. van Nostrand Co., 1963.

19 UNIVAC. 1100 Series. STAT PACK.

20 UNIVAC. 1100 Series. FORTRAN V. Programmer's Reference Manual.

21 UNIVAC. 1100 Series. EXEC II and 8. ASSEMBLFR. Programmer's Reference Manual.

22 Wilkes S.S. "Mathematical Statistics". New York, John Wiley and Sons. 1962.

APPENDIX A

TABLES

TABLE 1

SELECTED VALUES OF THE CHI-SQUARE DISTRIBUTION

	p=99%	p=95%	p=75%	p=50%	p=25%	p=5%	p=1%
ν= 1	0.00016	0.00393	0.1015	0.4549	1.323	3.841	6.653
ν= 2	0.00201	0.1026	0.5753	1.386	2.773	5.991	9.210
ν= 3	0.1148	0.3518	1.213	2.366	4.108	7.815	11.34
ν= 4	0.2971	0.7107	1.923	3.357	5,385	9.488	13.28
ν= 5	0.5543	1.1455	2.675	4.351	6.626	11.07	15.09
ν= 6	0.8720	1.635	3.455	5.348	7.841	12.59	16.81
ν= 7	1.239	2.167	4.255	6.346	9.037	14.07	18.48
ν= 8	1.646	2.733	5.071	7.344	10.22	15.51	20.09
ν= 9	2.088	3.325	5.899	8.343	11.39	16.92	21.67
ν=10	2.558	3.940	6.737	9.342	12.55	18.31	23.21
ν=11	3.053	4.575	7.584	10.34	13.70	19.68	24.73
ν=12	3.571	5.226	8.438	11.34	14.84	21.03	26.22
ν=15	5.229	7.261	11.04	14.34	18,25	25.00	30.58
ν=20	8.260	10.85	15.45	19.34	23.83	31.41	37.57
ν=30	14.95	18.49	24.48	29.34	34.80	43.77	50.89
ν=50	29.71	34.76	42.94	49.33	56.33	67.50	76.15
ν=30	approximately $\nu + 2\sqrt{\nu}x_p + \frac{4}{3}x_p^2 - \frac{2}{3}$						
x_p	-2.33	-1.64	-.675	0.00	0.675	1.64	2.33

TABLE 2

SELECTED VALUES OF THE DISTRIBUTION OF K_n^+ AND K_n^-

The approximations for n > 30 are only conjectures which have not been proved, except when n = ∞

	p=99%	p=95%	p=75%	p=50%	p=25%	p=5%	p=1%
n= 1	0.01000	0.05000	0.2500	0.5000	0.7500	0.9500	0.9900
n= 2	0.01400	0.06749	0.2929	0.5176	0.7071	1.0980	1.2728
n= 3	0.01699	0.07919	0.3112	0.5147	0.7539	1.1017	1.3589
n= 4	0.01943	0.08789	0.3202	0.5110	0.7642	1.1304	1.3777
n= 5	0.02152	0.09471	0.3249	0.5245	0.7674	1.1392	1.4024
n= 6	0.02336	0.1002	0.3272	0.5319	0.7703	1.1463	1.4144
n= 7	0.02501	0.1048	0.3280	0.5364	0.7755	1.1537	1.4246
n= 8	0.02650	0.1086	0.3280	0.5392	0.7797	1.1586	1.4327
n= 9	0.02786	0.1119	0.3274	0.5411	0.7825	1.1624	1.4388
n=10	0.02912	0.1147	0.3297	0.5426	0.7845	1.1658	1.4440
n=11	0.03028	0.1172	0.3330	0.5439	0.7863	1.1688	1.4484
n=12	0.03137	0.1193	0.3357	0.5453	0.7880	1.1714	1.4521
n=15	0.03424	0.1244	0.3412	0.5500	0.7926	1.1773	1.4606
n=20	0.03807	0.1298	0.3461	0.5547	0.7975	1.1839	1.4698
n=30	0.04354	0.1351	0.3509	0.5605	0.8036	1.1916	1.4801
n>30	$0.07080 - \frac{0.15}{\sqrt{n}}$	$0.1601 - \frac{0.14}{\sqrt{n}}$	$0.3793 - \frac{0.15}{\sqrt{n}}$	$0.5887 - \frac{0.15}{\sqrt{n}}$	$0.8326 - \frac{0.16}{\sqrt{n}}$	$1.2239 - \frac{0.17}{\sqrt{n}}$	$1.5174 - \frac{0.20}{\sqrt{n}}$

TABLE 3

SAMPLE RESULTS OF THE SPECTRAL TEST

	a	m	C_2	C_3	C_4
1	23	$10^8 + 1$	0.000017	0.00051	0.014
2	$2^7 + 1$	2^{35}	0.000002	0.00026	0.040
3	$2^{18} + 1$	2^{35}	3.14	0.000000002	0.000000003
4	3141592653	2^{35}	0.27	0.13	0.11
5	3141592221	10^{10}	1.35	0.06	4.67
6	3141592421	10^{10}	2.69	0.35	0.54
7	3141592621	10^{10}	1.44	0.43	1.91
8	3141592821	10^{10}	0.16	2.90	0.34
9	3141592221	2^{35}	1.24	1.69	1.11
10	3141592621	2^{35}	3.02	0.17	1.25
11	2718281821	2^{35}	2.59	1.15	1.75
12	$2^{23} + 2^{12} + 5$	2^{35}	0.015	2.78	0.066
13	$2^{23} + 2^{13} + 5$	2^{35}	0.015	1.48	0.066
14	$2^{23} + 2^{14} + 5$	2^{35}	1.12	1.66	0.066
15	$2^{22} + 2^{13} + 5$	2^{35}	0.75	0.30	0.066
16	$2^{24} + 2^{13} + 5$	2^{35}	0.0008	2.92	0.066
17	5^{13}	2^{35}	3.03	0.61	1.84
18	5^{15}	2^{35}	2.02	4.12	4.04

APPENDIX B

FLOWCHARTS

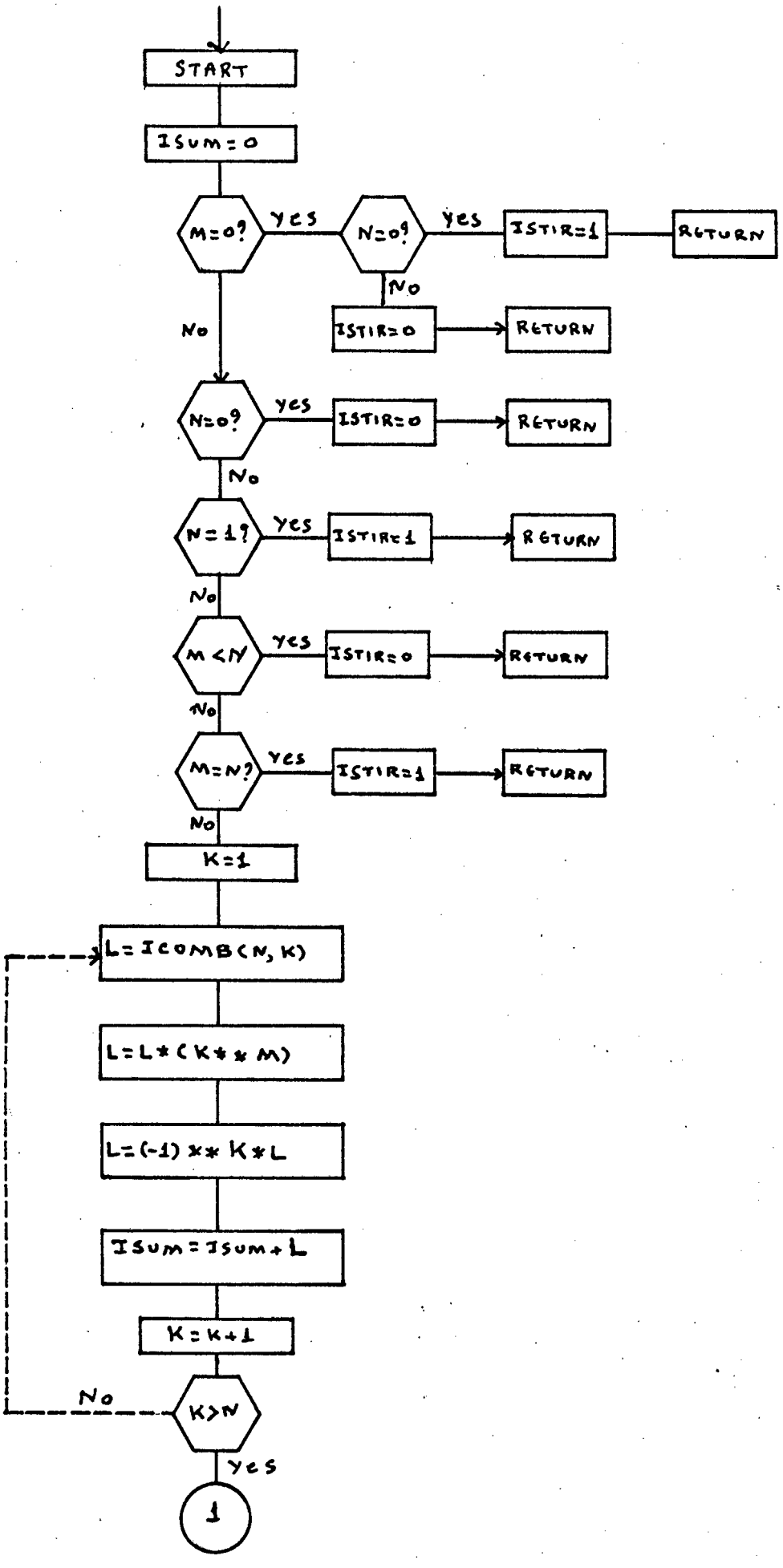


Figure Flowchart of function ISTIR.

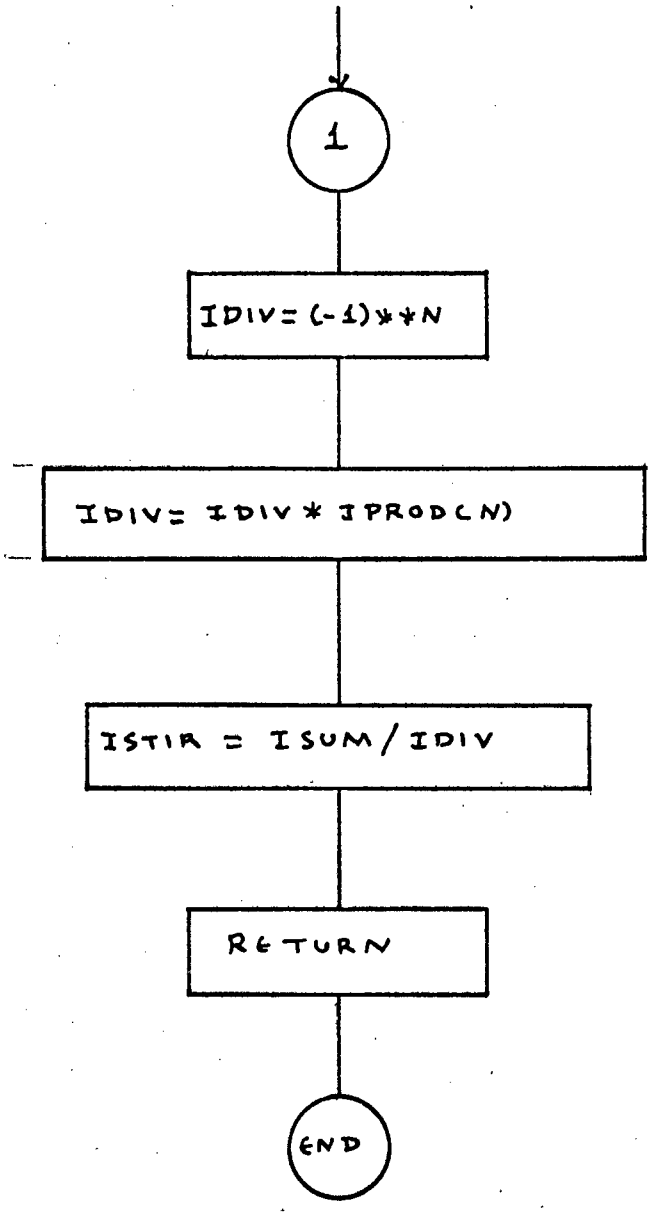


Figure (continued).

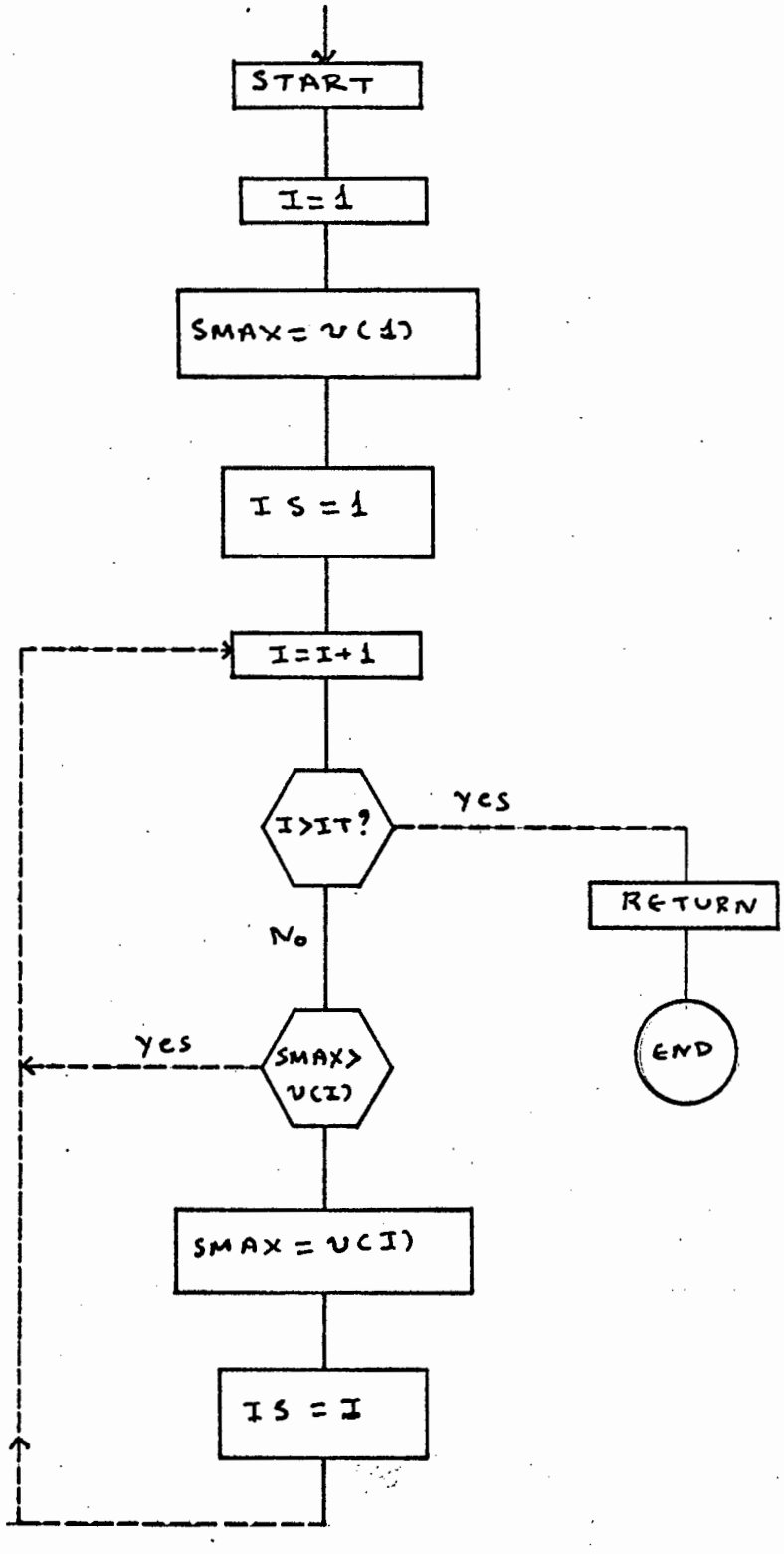


Figure Flowchart of subroutine MAXIM.

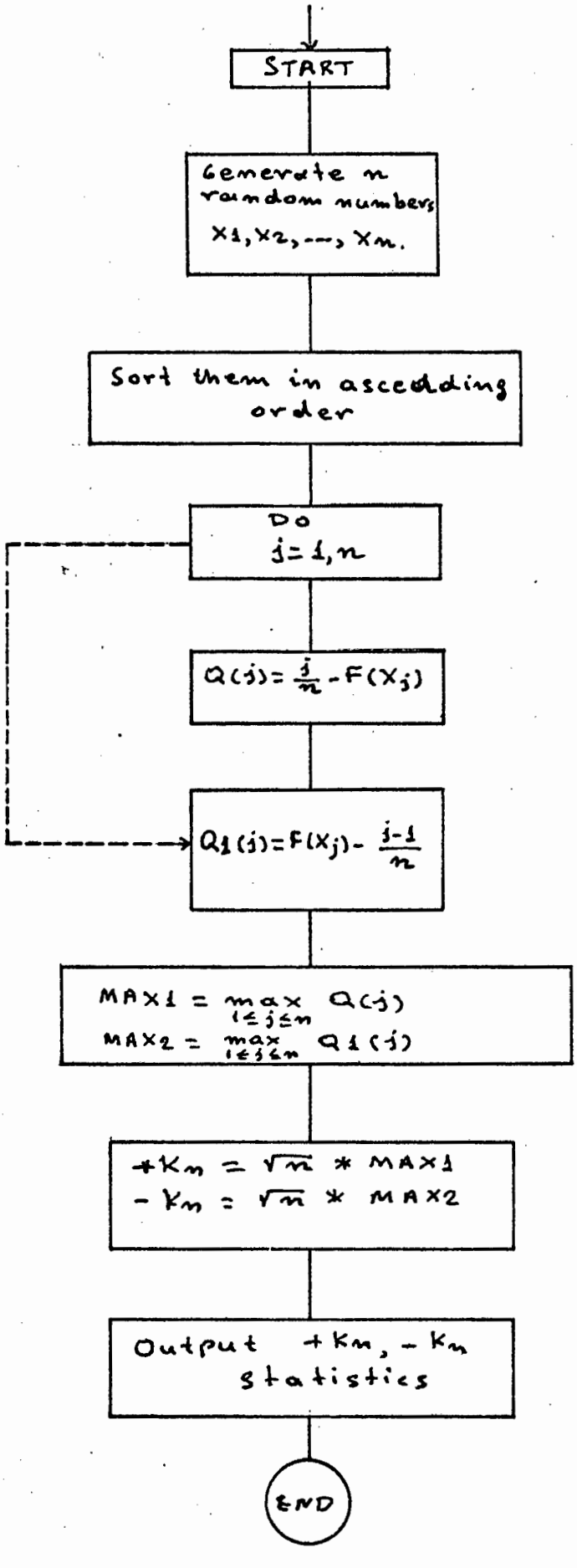


Figure Flowchart of program TEST3.

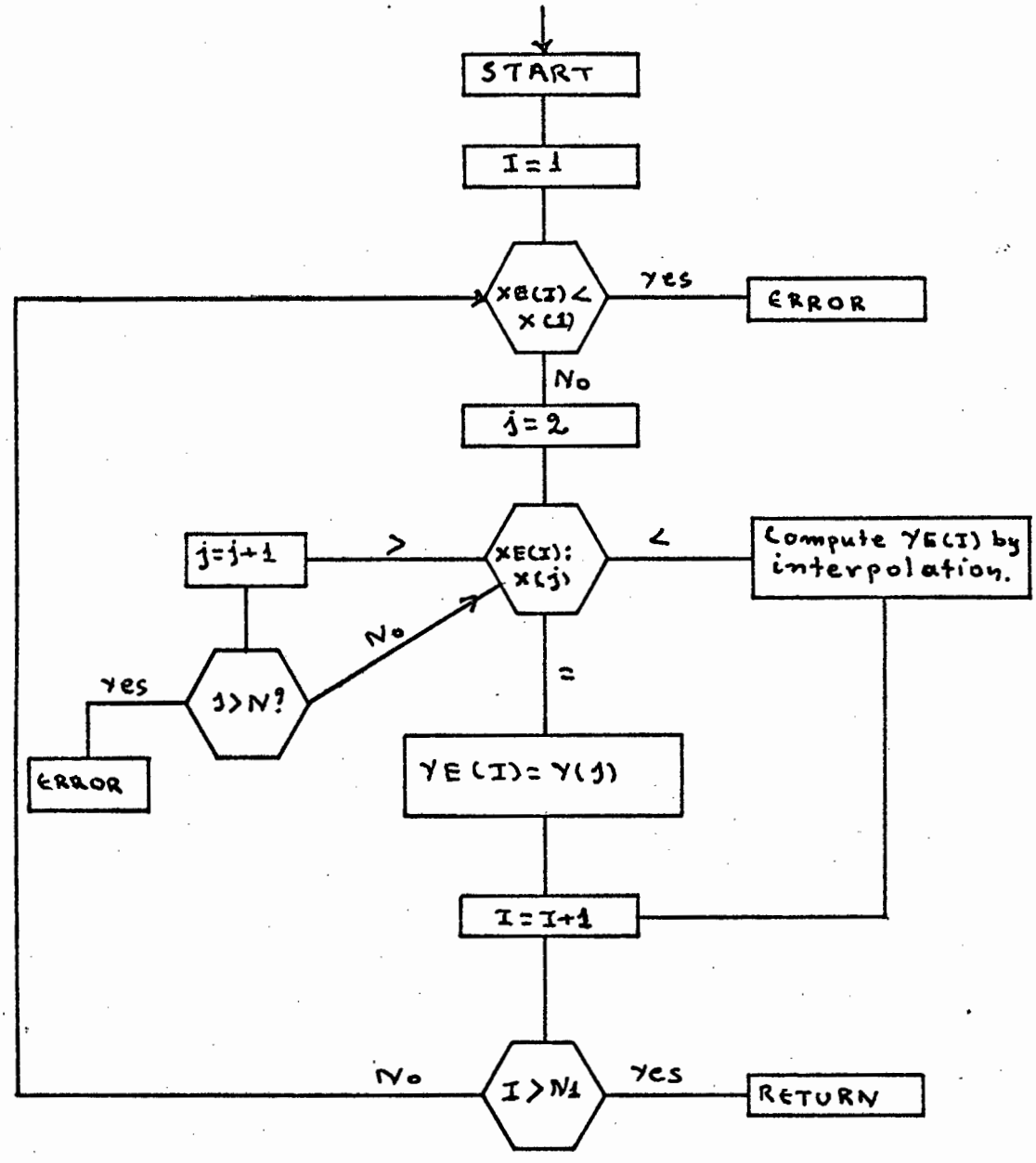


Figure Flowchart of subroutine INTER.

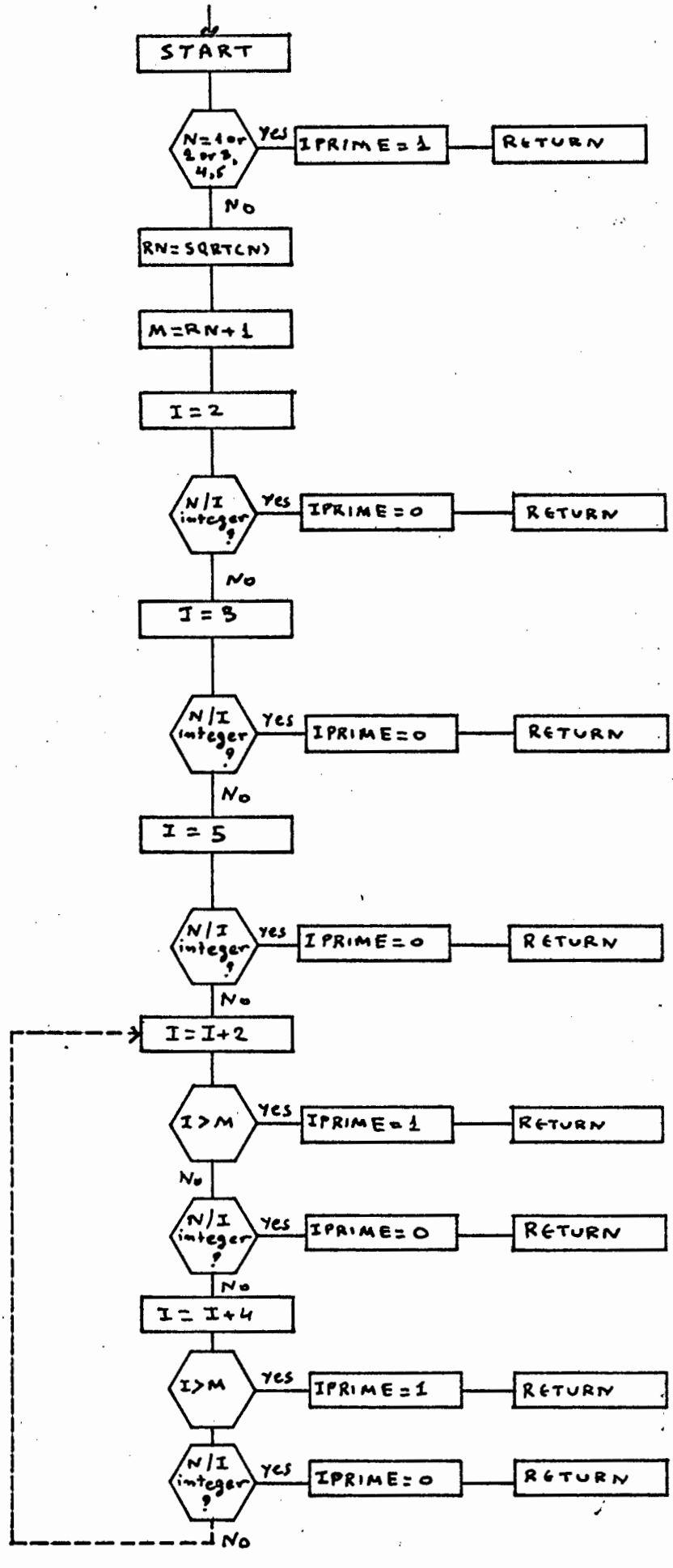


Figure Flowchart of function IPRIME.

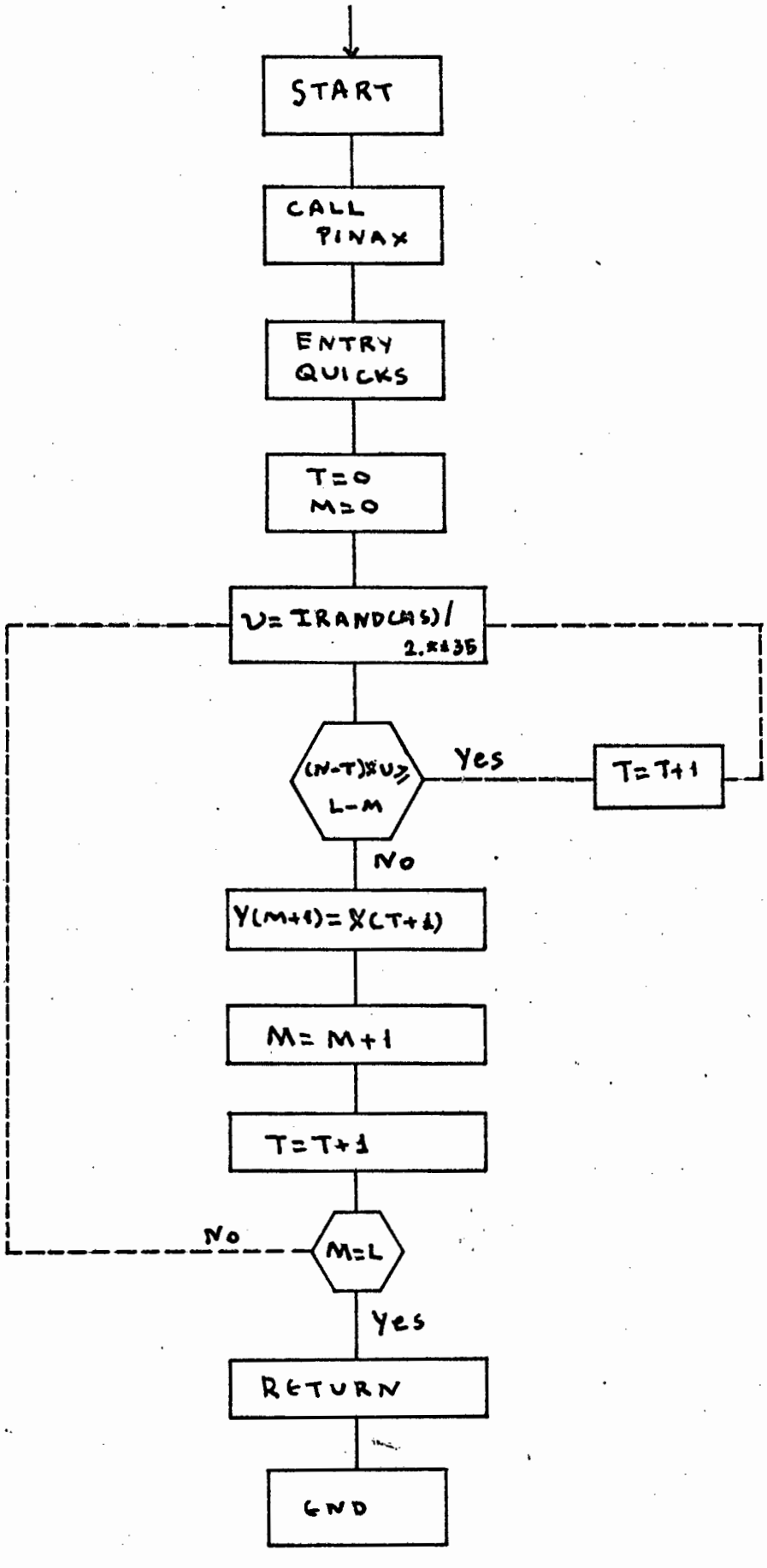


Figure Flowchart of subroutine SAMPLE.

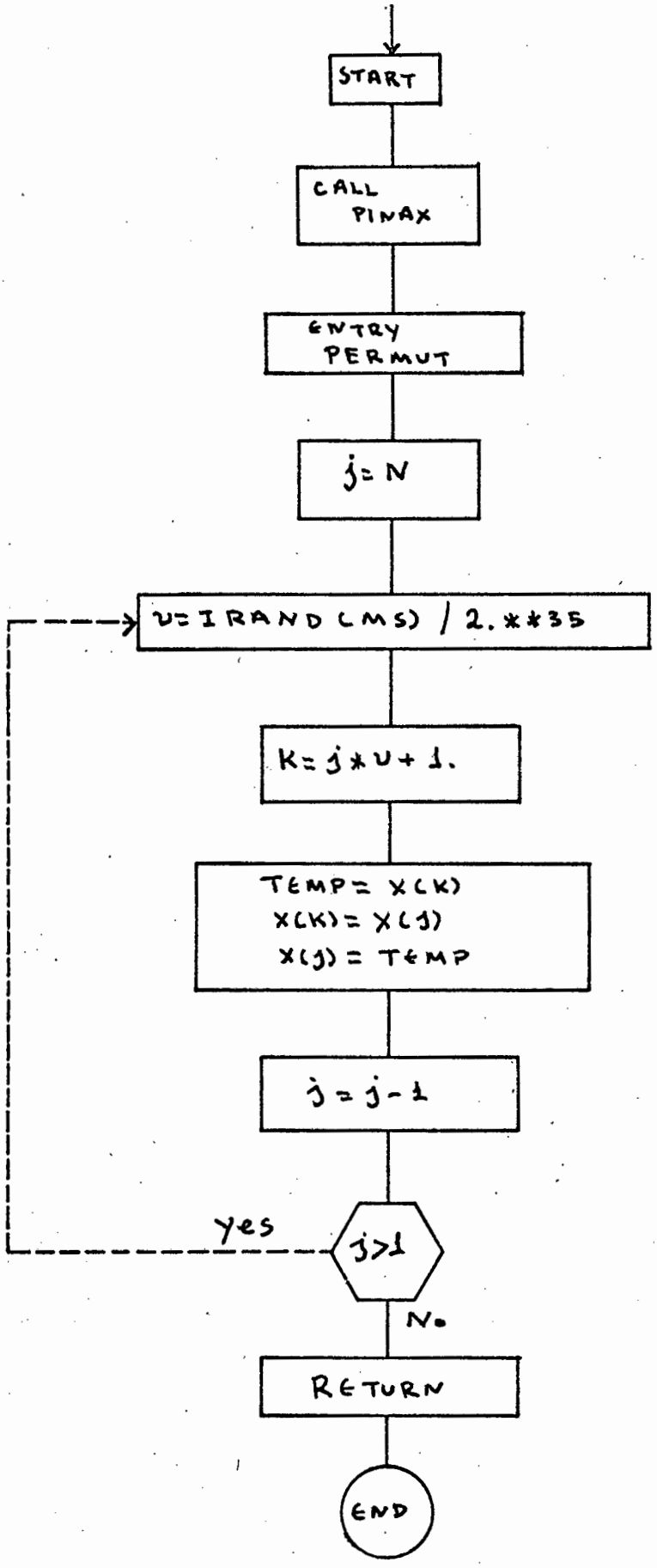


Figure Flowchart of subroutine SHUFFL

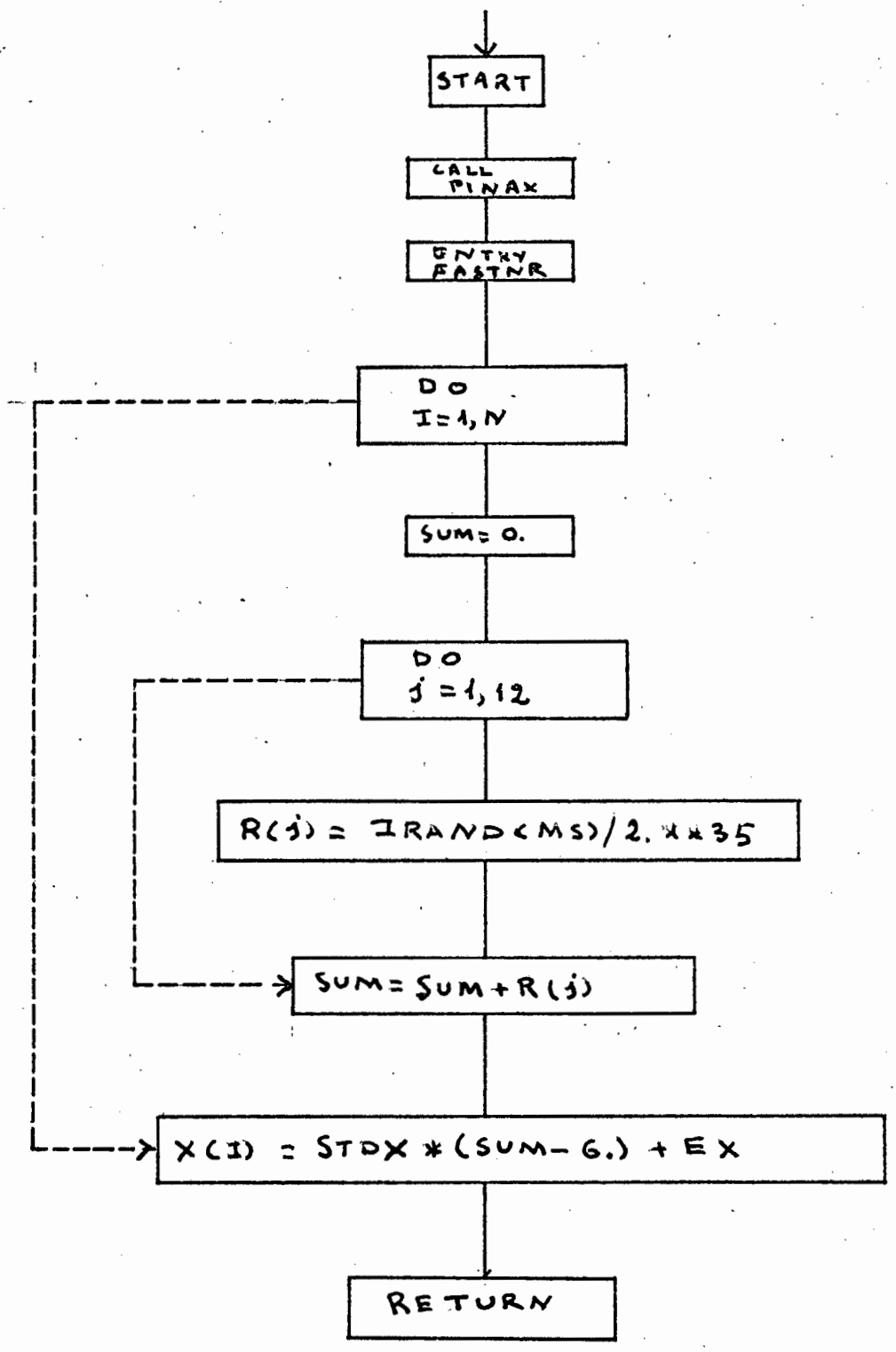


Figure Flowchart of subroutine RANDNR.

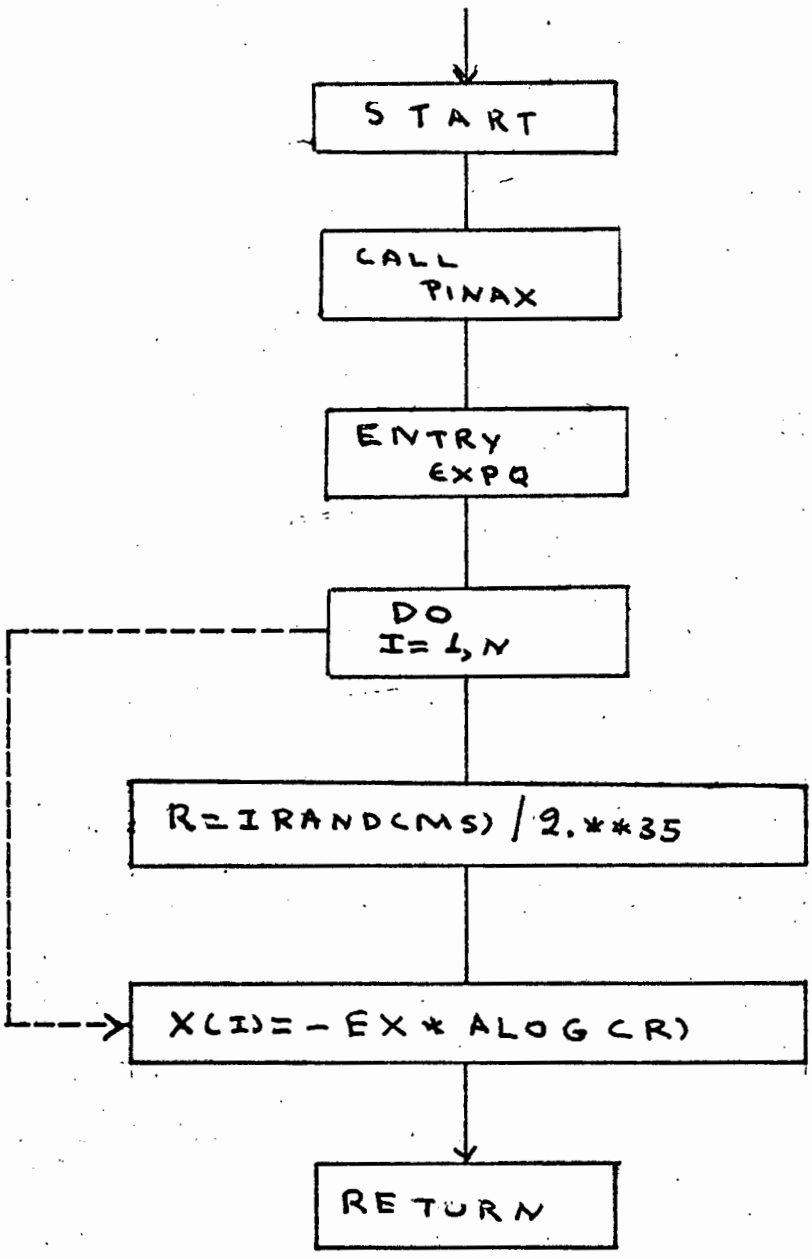


Figure Flowchart of subroutine EXPR.

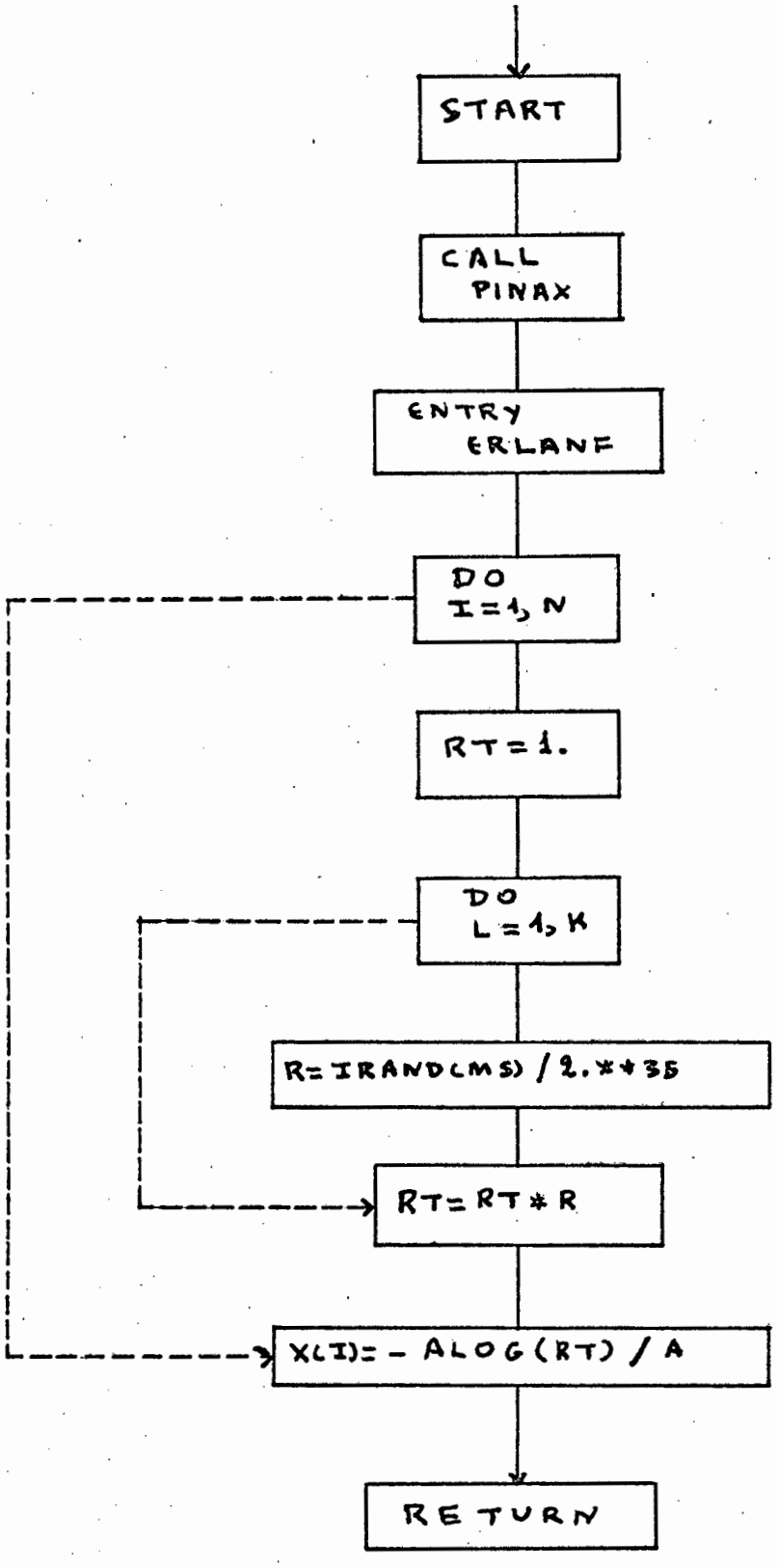


Figure Flowchart of subroutine ERLANG.

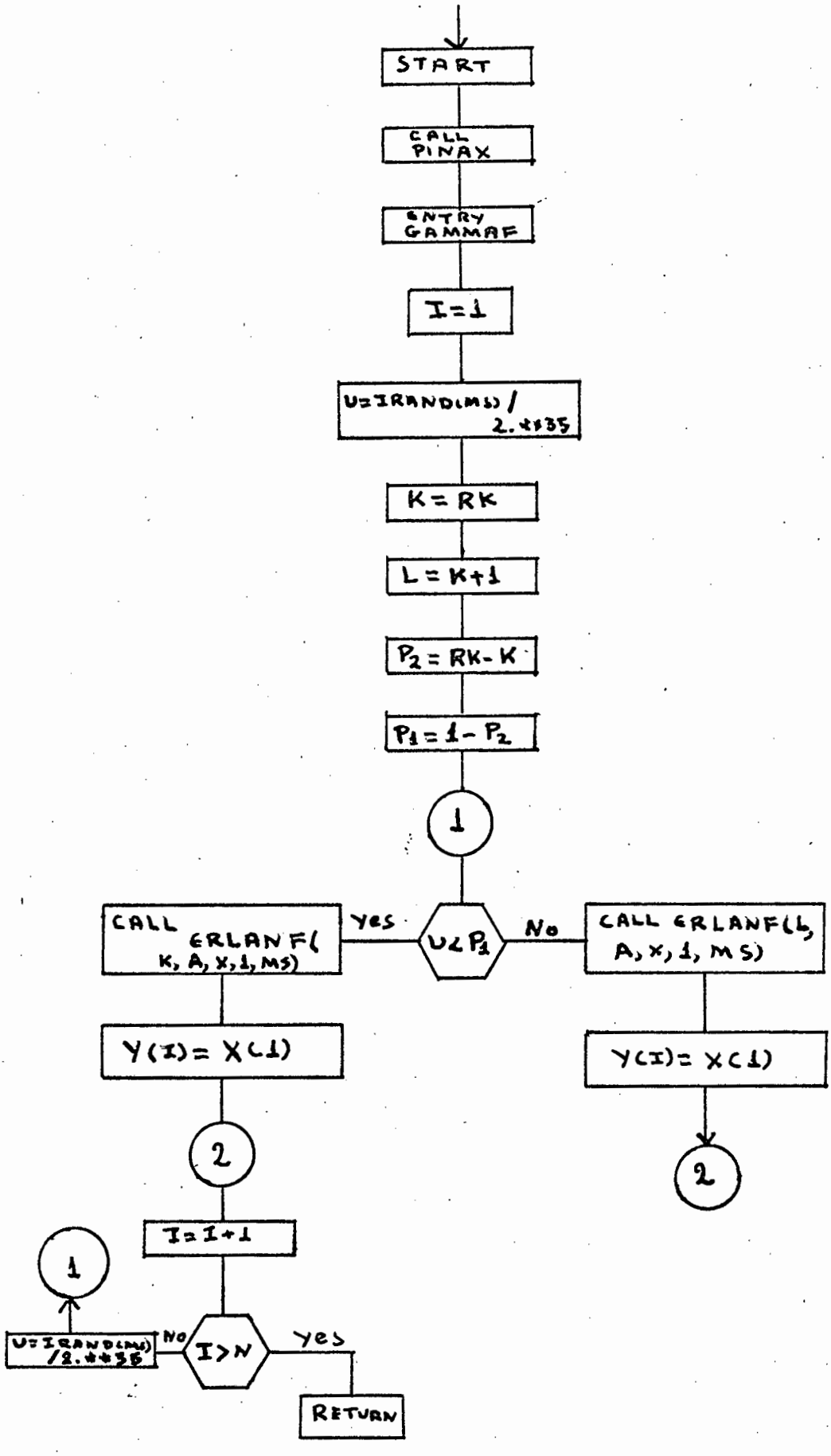


Figure Flowchart of subroutine GAMMA.

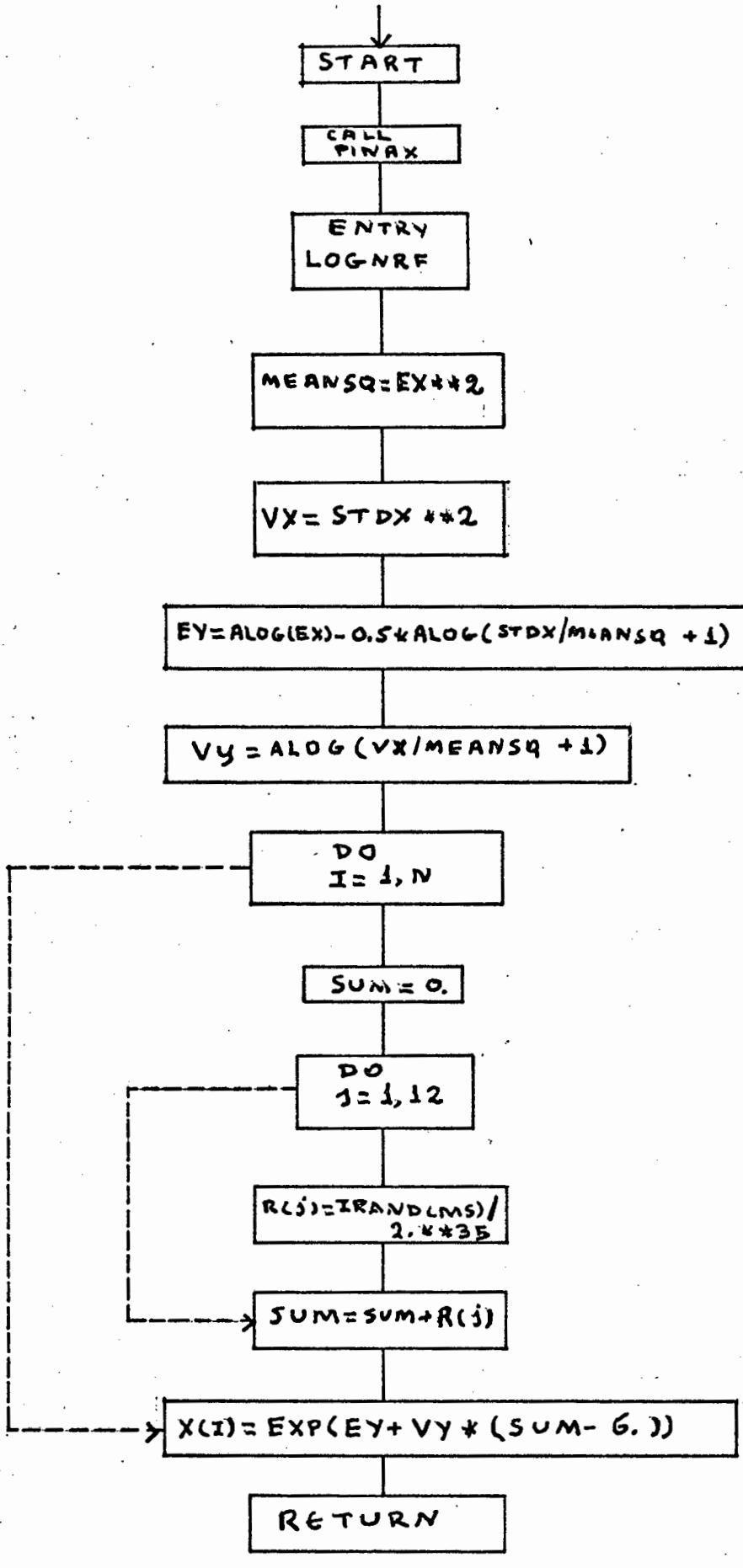


Figure Flowchart of subroutine LOGNOR.

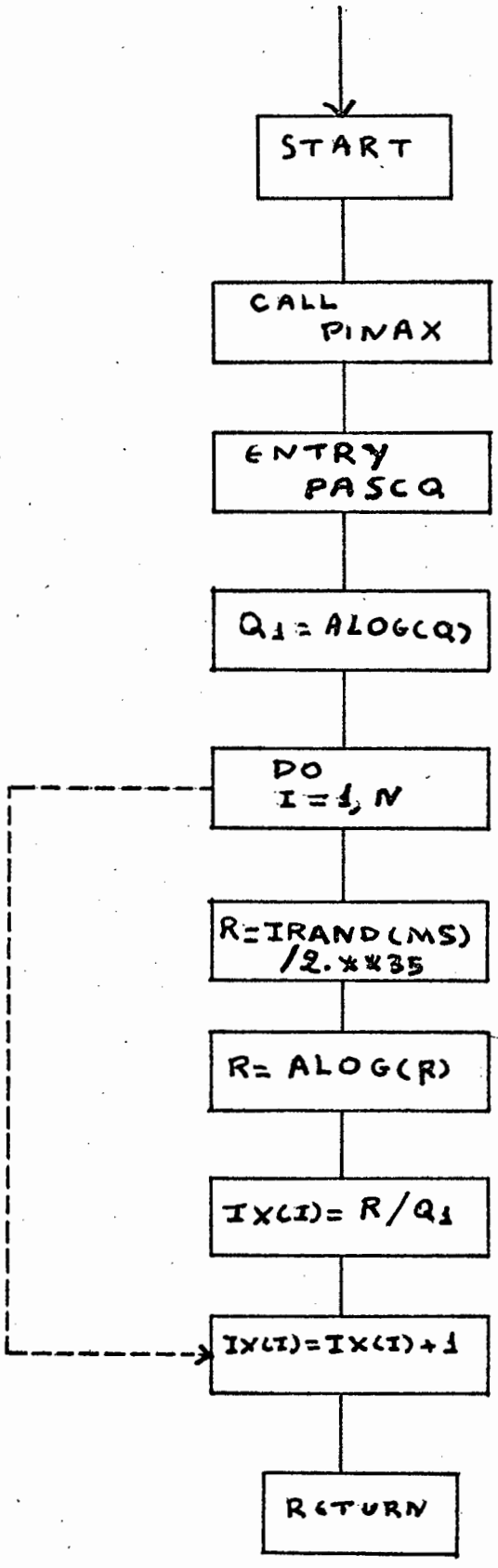


Figure Flowchart of subroutine PASCAL.

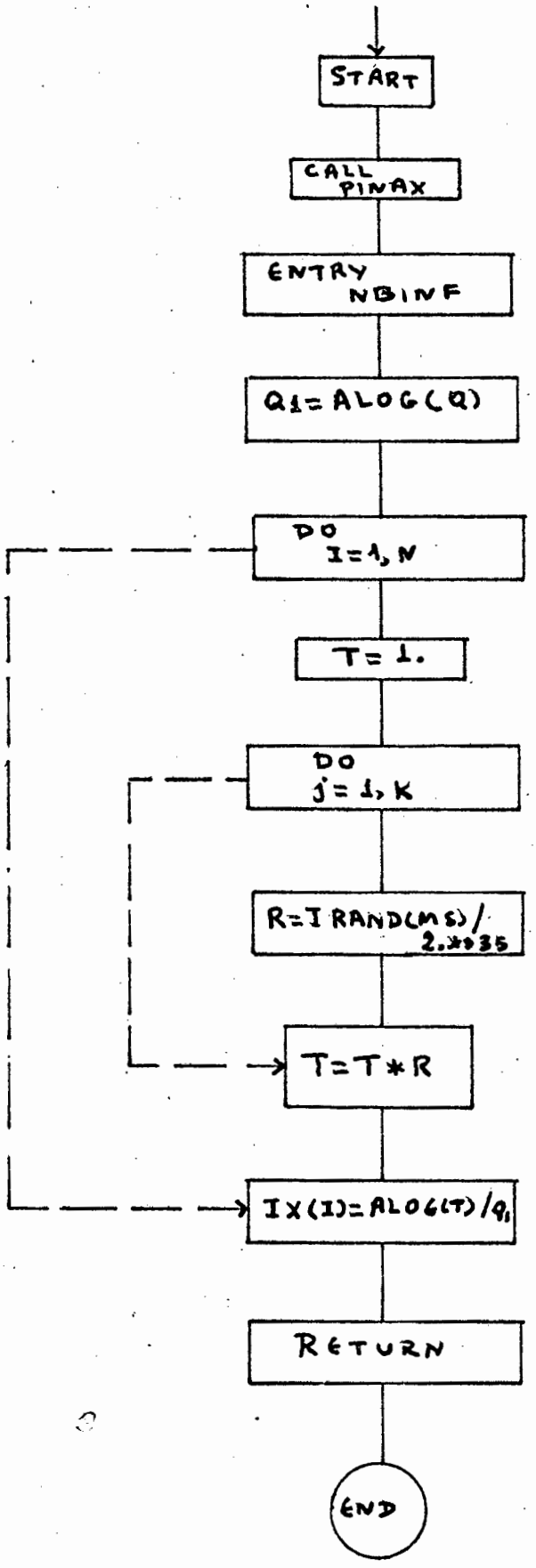


Figure . Flowchart of subroutine NBIN.

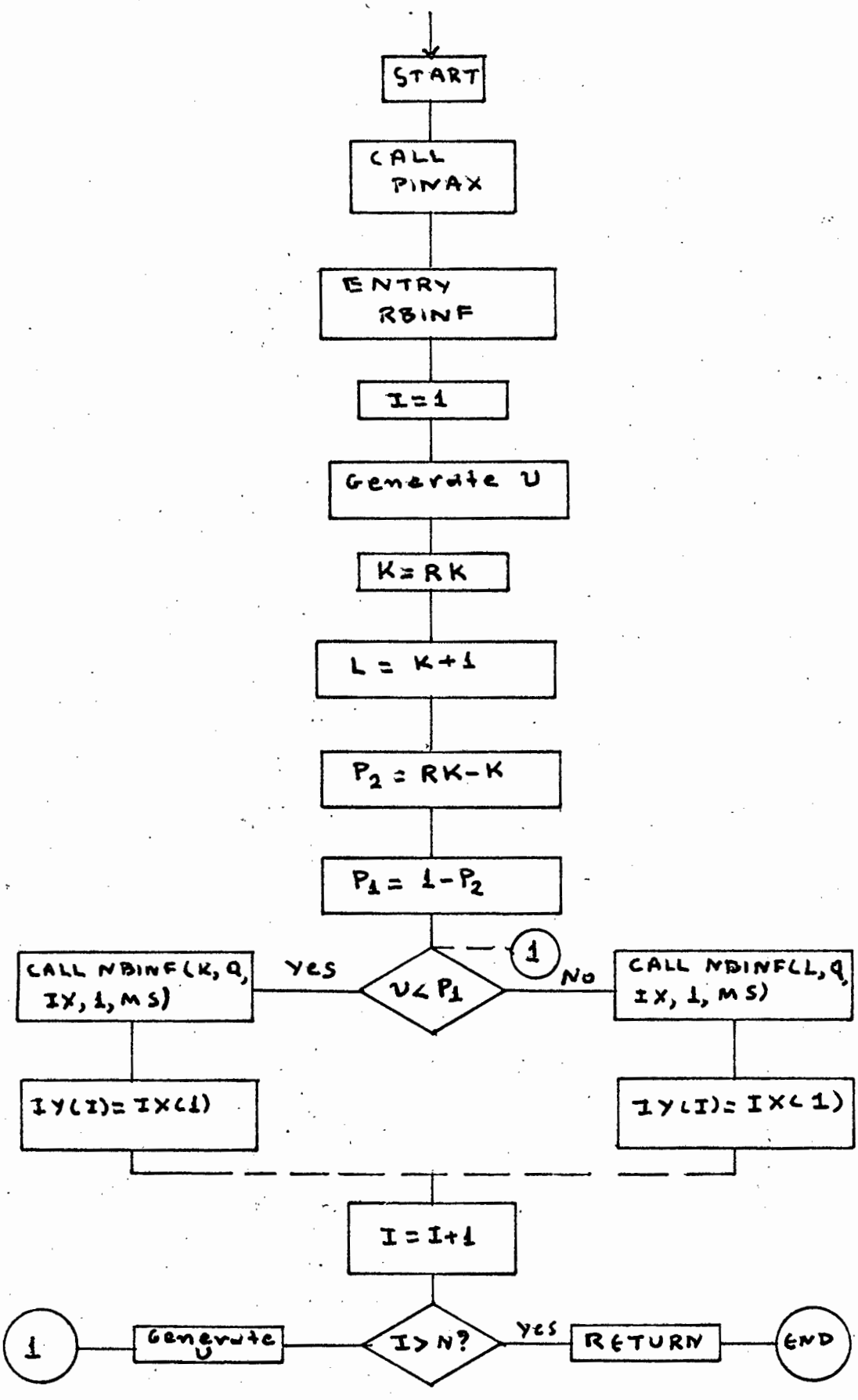


Figure Flowchart of sub. RBIN.

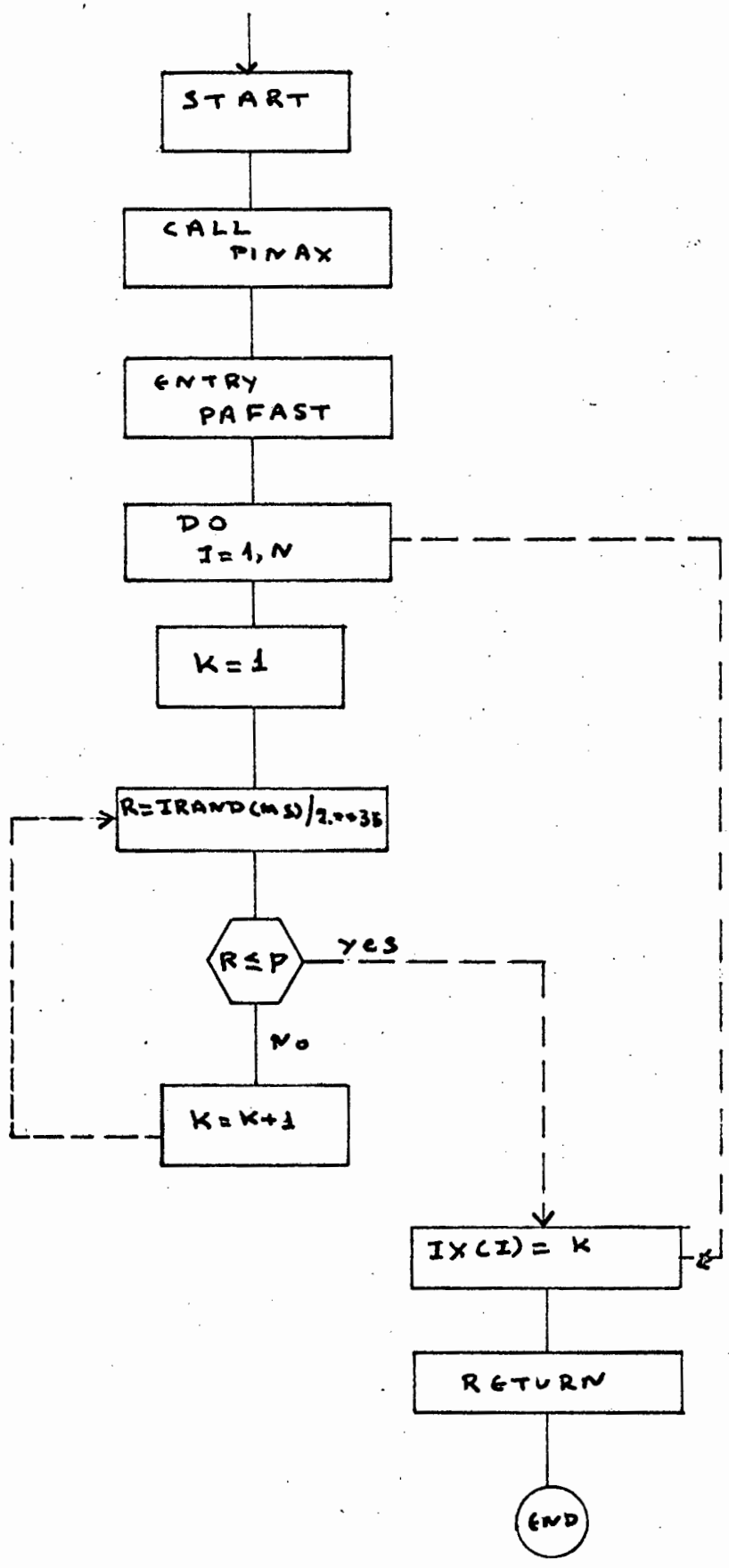


Figure . Flowchart of subroutine PASBIG.

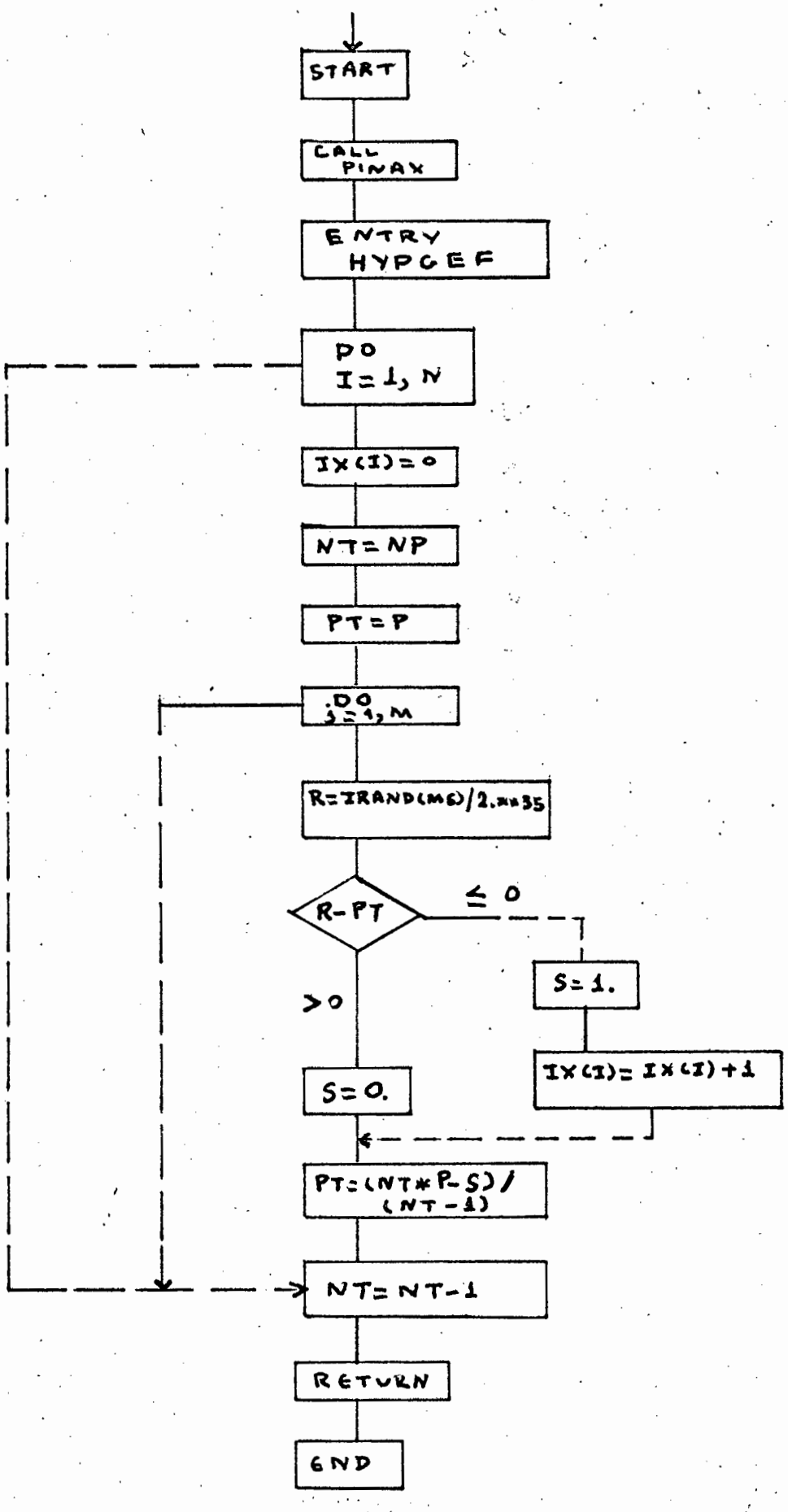


Figure Flowchart of sub. HYPGEO.

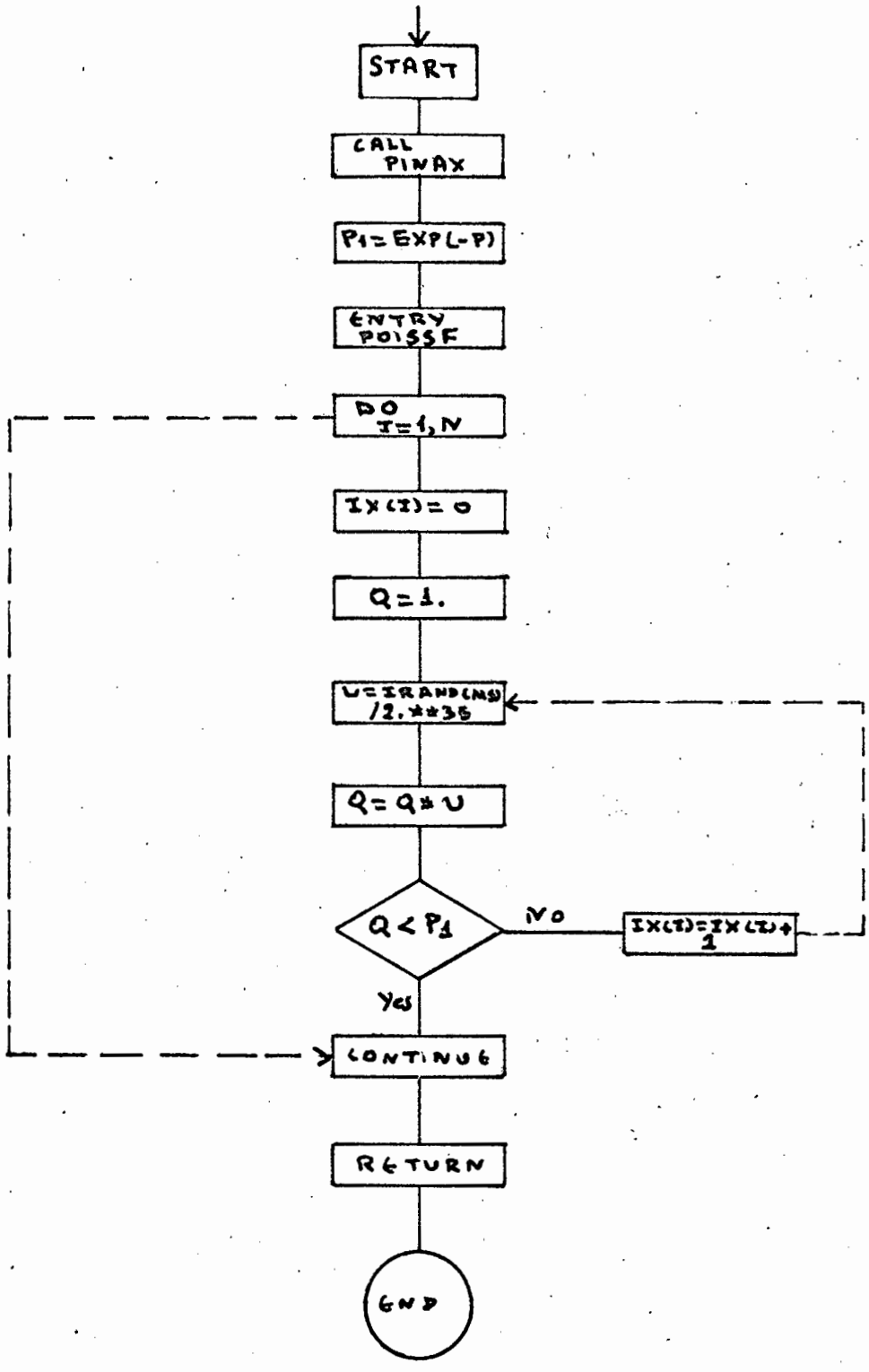


Figure Flowchart of subroutine POISSN.

APPENDIX C

FORTRAN LISTINGS AND OUTPUT RESULTS

```
*AD.RANDSUB2
S(1)    RES     0
        AXR*
X15     EQU     15
SETP*   LA      A1,*0,X11       .ACC1=MULTIPLIER.
        SA      A1,MLTN         .MLTN=ACC1.
        LA      A1,*1,X11       .ACC1=INCREMENT.
        SA      A1,INCNA        .INCNA=ACC1.
        J       3,X11           .RETURN.
DINAX*  SX      X1,SAVX1        .
        LR,XU   R1,127          .SET LOOP COUNT TO 127.
        LX,XU   X1,0            .I=0.
        LA      A1,*0,X11       .ACC1=X(0).
AGAIN   MI      A1,MLTN         .(ACC1,ACC2)=MLTN*X(I).
        DA      A1,INCN         .(ACC1,ACC2)=MLTN*X(I)+INCN.
        AND     A2,(037777777777777) .ACC3=(MLTN*X(I)+INCN)MOD2**35.
        SA      A3,V,X1         .V(I+1)=X(I+1).
        LA      A1,A3           .ACC1=X(I+1).
        AX,XU   X1,1            .I=I+1.
        JGD     R1,AGAIN        .STOP LOOP WHEN 128 NOS. GENERATED.
        SA      A1,*0,X11       .STORE X(127) IN 1ST PAR. OF CALL. SEQ.
        LX      X1,SAVX1        .
        J       2,X11           .RETURN.
IRAND*  LA      A1,*0,X11       .ACC1=X(I).
        LA      A5,A1           .ACC5=X(I).
        MI      A1,MLTN         .(ACC1,ACC2)=MLTN*X(I).
        DA      A1,INCN         .(ACC1,ACC2)=MLTN*X(I)+INCN.
        AND     A2,(037777777777777) .ACC3=(MLTN*X(I)+INCN)MOD2**35.
        SA      A3,*0,X11       .ACC3=X(I+1).
        LA      A3,A5           .ACC3=X(I).
        SSL     A3,28           .X15=K WHERE K=INT.PART Y DEV.BY 2**28.
        LA      A0,V,X15        .IRAND FUNCTION VALUE=V(K+1).
        LA      A1,*0,X11       .ACC1=X(I+1).
        SA      A1,V,X15        .V(K+1)=X(I+1).
        J       2,X11           .RETURN.
.
.       DATA    AREA            .
.
S(0)    RES     0
MLTN    +       27102818211     .
INCN    +       0               .
INCNA   +       72610677265     .
V       RES     128             .
SAVX1   +       0               .
        END
```

```
0:03-CHEK(0)
01:           DIMENSION IX(30000)
02:           WRITE(5,11)
03:    11     FORMAT(1H1,'OUTPUT FROM IRAND'/)
04:           MS=113
05:           CALL PINAX(MS)
06:           DO 1 I=1,30000
07:    1      IX(I)=IRAND(MS)
10:           WRITE(5,10) IX(30000)
11:    10     FORMAT(10X,'THE LAST NUMBER GENERATED IS',I12)
12:           END
```

```
 OUTPUT FROM IRAND

          THE LAST NUMBER GENERATED IS  9622815040
```

```
 NORMAL EXIT.  EXECUTION TIME:          1713 MILLISECONDS.
```

```
09=CHEK(0)
:         DIMENSION IX(30000)
:         WRITE(5,11)
:   11    FORMAT(1H1,'OUTPUT FROM MRAND'/)
:         MS=113
:         NT=MS
:         CALL TABLE(MS)
:         DO 1 I=1,30000
:   1     IX(I)=MRAND(MS,NT)
:         WRITE(5,10) IX(30000)
:   10    FORMAT(10X,'THE LAST NUMBER GENERATED IS',I12)
:         END
```

```
 OUTPUT FROM MRAND

          THE LAST NUMBER GENERATED IS 22668474579
```

```
 NORMAL EXIT.  EXECUTION TIME:        2637 MILLISECONDS.

 @FIN
```

```
)*PAO.TEST1
 1              DIMENSION L(64)
 2              READ(8,5) NTEST
 3        5     FORMAT(I5)
 4              WRITE(5,7) NTEST
 5        7     FORMAT(1H1,12X,'TEST1(FIRST TEST OF IRAND) TESTS WHETHER
 6             1OUTPUT FROM'/13X,'RAND BEHAVES AS IF DRAWN FROM A UNIFORMLY
 7             2DISTIBUTED POPULATION.'/13X,'RESULTS OBTAINED FROM EXECUTING
 8             3THIS TEST',I4,'TIMES ARE GIVEN BELOW.')
 9              WRITE(5,11)
10       11    FORMAT(1H1,'OUTPUT  FROM  IRAND'//)
11       C  THE ITERATION SCHEME BEGINS.
12              DO150NTS=1,NTEST
13       C READ THE INPUT VALUES. **NN** IS THE NUMBER OF RANDOM NUMBERS
14       C IN THE GROUP. **NG** IS THE NUMBER OF GROUPS.
15              READ(8,10)  NN,MS,NG
16       10     FORMAT(4X,I4,I12,I2)
17              MSTART=MS
18       C **NF** DENOTES THE NUMBER OF DEGREES OF FREEDOM.
19              NF=NG-1
20              CALL PINAX(MS)
21              NB=2**34
22              NB=NB/NG
23              DO 17I=1,NG
24       17     L(I)=0
25              DO 20 K=1,NN
26              J=IRAND(MS)
27       C INTEGER J IN THE RANGE 0,2**35-1 IS CONVERTED TO AN INTEGER
28       C BETWEEN 1 AND NG.
29              I=(J/(2*NB))+1
30       20     L(I)=L(I)+1
31              K=0
32       C CALCULATION OF THE CHI SQUARE STATISTIC.
33              DO 30I=1,NG
34        30    K=K+L(I)**2
35              V=K*NG
36              U=NN
37              V=(V/U)-U
38              VAR=CHI(V,NF,$75)
39              VAR=VAR*100
40              VAR=100.-VAR
41              WRITE(5,40) NTS,NN,NG,MSTART,V,NF,VAR
42        40    FORMAT(8X,'TEST',1X,I3,'NO. IN',1X,I4,'GROUPS'1X,I2,10X,'START.
43             1NUMBER='I12/8X,'CHI SQUARE STATISTIC=',F13.3,2X,'DEG. OF FREEC='
44             2,1X,I4,')'/8X,'STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE',
45             31X,F8.2,'PER CENT OF THE TINE.'/)
46              GO TO 150
47        75    WRITE(5,80)
48       80     FORMAT(1H1,'OVERFLOW IN CHI')
49       150    CONTINUE
50              END
```

OUTPUT FROM IRAND

```
TEST   1NO. IN  100GROUPS  5          STARTINGNUMBER=         1111
CHI SQUARE STATISTIC=       9.200  DEG. OF FREEDOM=    4)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE     5.63PER CENT OF THE TINE.

TEST   2NO. IN  100GROUPS  5          STARTINGNUMBER=         1112
CHI SQUARE STATISTIC=       1.600  DEG. OF FREEDOM=    4)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    80.88PER CENT OF THE TINE.

TEST   3NO. IN  100GROUPS  5          STARTINGNUMBER=         1113
CHI SQUARE STATISTIC=       4.600  DEG. OF FREEDOM=    4)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    33.09PER CENT OF THE TINE.

TEST   4NO. IN  100GROUPS  5          STARTINGNUMBER=   2468134327
CHI SQUARE STATISTIC=       5.300  DEG. OF FREEDOM=    4)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    25.79PER CENT OF THE TINE.

TEST   5NO. IN  100GROUPS  5          STARTINGNUMBER=   2468135790
CHI SQUARE STATISTIC=       5.800  DEG. OF FREEDOM=    4)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    21.46PER CENT OF THE TINE.

TEST   6NO. IN 1000GROUPS 50          STARTINGNUMBER=          928
CHI SQUARE STATISTIC=      44.800  DEG. OF FREEDOM=   49)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    64.40PER CENT OF THE TINE.

TEST   7NO. IN 1000GROUPS 50          STARTINGNUMBER=          929
CHI SQUARE STATISTIC=      55.600  DEG. OF FREEDOM=   49)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    24.02PER CENT OF THE TINE.

TEST   8NO. IN 1000GROUPS 50          STARTINGNUMBER=          930
CHI SQUARE STATISTIC=      46.700  DEG. OF FREEDOM=   49)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    56.69PER CENT OF THE TINE.

TEST   9NO. IN 1000GROUPS 50          STARTINGNUMBER=          931
CHI SQUARE STATISTIC=      56.000  DEG. OF FREEDOM=   49)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    22.89PER CENT OF THE TINE.

TEST  10NO. IN 1000GROUPS 50          STARTINGNUMBER=   2468134325
CHI SQUARE STATISTIC=      55.300  DEG. OF FREEDOM=   49)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    24.90PER CENT OF THE TINE.
```

OUTPUT FROM MRAND

```
TEST   1NO. IN  100GROUPS  5          STARTINGNUMBER=        1111
CHI SQUARE STATISTIC=      3.100  DEG. OF FREEDOM=    4)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    54.12PER CENT OF THE TINE.

TEST   2NO. IN  100GROUPS  5          STARTINGNUMBER=        1112
CHI SQUARE STATISTIC=      6.200  DEG. OF FREEDOM=    4)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    18.47PER CENT OF THE TINE.

TEST   3NO. IN  100GROUPS  5          STARTINGNUMBER=        1113
CHI SQUARE STATISTIC=      1.100  DEG. OF FREEDOM=    4)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    89.43PER CENT OF THE TINE.

TEST   4NO. IN  100GROUPS  5          STARTINGNUMBER=  2468134327
CHI SQUARE STATISTIC=      1.700  DEG. OF FREEDOM=    4)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    79.07PER CENT OF THE TINE.

TEST   5NO. IN  100GROUPS  5          STARTINGNUMBER=  2468135790
CHI SQUARE STATISTIC=      4.300  DEG. OF FREEDOM=    4)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    36.69PER CENT OF THE TINE.

TEST   6NO. IN 1000GROUPS 50          STARTINGNUMBER=         928
CHI SQUARE STATISTIC=     57.200  DEG. OF FREEDOM=   49)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    19.69PER CENT OF THE TINE.

TEST   7NO. IN 1000GROUPS 50          STARTINGNUMBER=         929
CHI SQUARE STATISTIC=     50.800  DEG. OF FREEDOM=   49)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    40.25PER CENT OF THE TINE.

TEST   8NO. IN 1000GROUPS 50          STARTINGNUMBER=         930
CHI SQUARE STATISTIC=     39.300  DEG. OF FREEDOM=   49)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    83.76PER CENT OF THE TINE.

TEST   9NO. IN 1000GROUPS 50          STARTINGNUMBER=         931
CHI SQUARE STATISTIC=     28.000  DEG. OF FREEDOM=   49)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE    99.31PER CENT OF THE TINE.

TEST  10NO. IN 1000GROUPS 50          STARTINGNUMBER=  2468134325
CHI SQUARE STATISTIC=     63.300  DEG. OF FREEDOM=   49)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE     8.23PER CENT OF THE TINE.
```

```
0*PAO.TESTA
      DIMENSION L(64)
      READ(8,5) NTEST
    5 FORMAT(I5)
      WRITE(5,7) NTEST
    7 FORMAT(1H1,12X,'TEST1(FIRST TEST OF IRAND) TESTS WHETHER
     1OUTPUT FROM'/13X,'RAND BEHAVES AS IF DRAWN FROM A UNIFORMLY
     2DISTIBUTED POPULATION.'/13X,'RESULTS OBTAINED FROM EXECUTING
     3THIS TEST',I4,'TIMES ARE GIVEN BELOW.')
      WRITE(5,11)
   11 FORMAT(1H1,'OUTPUT  FROM  MRAND'//)
      N1=7261067265
      CALL SETM(2718281821,N1,3141592653,N1)
C  THE ITERATION SCHEME BEGINS.
      DO150NTS=1,NTEST
C READ THE INPUT VALUES. **NN** IS THE NUMBER OF RANDOM NUMBERS
C IN THE GROUP. **NG** IS THE NUMBER OF GROUPS.
      READ(8,10)  NN,MS,MS1,NG
   10 FORMAT(4X,I4,2I2,I2)
      MSTART=MS
C **NF** DENOTES THE NUMBER OF DEGREES OF FREEDOM.
      NF=NG-1
      CALL TABLE(MS)
      NB=2**34
      NB=NB/NG
      DO 17I=1,NG
   17 L(I)=0
      DO 20 K=1,NN
      J=MRAND(MS,MS1)
C INTEGER J IN THE RANGE 0,2**35-1 IS CONVERTED TO AN INTEGER
C BETWEEN 1 AND NG.
      I=(J/(2*NB))+1
   20 L(I)=L(I)+1
      K=0
C CALCULATION OF THE CHI SQUARE STATISTIC.
      DO 30I=1,NG
   30 K=K+L(I)**2
      V=K*NG
      U=NN
      V=(V/U)-U
      VAR=CHI(V,NF,S75)
      VAR=VAR*100
      VAR=100.-VAR
      WRITE(5,40) NTS,NN,NG,MSTART,V,NF,VAR
   40 FORMAT(8X,'TEST',1X,I3,'NO. IN',1X,I4,'GROUPS'1X,I2,10X,'STARTING
     1NUMBER='I12/9X,'CHI SQUARE STATISTIC=',F13.3,2X,'DEG. OF FREEDOM='
     2,1X,I4,')'/8X,'STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE',
     31X,F8.2,'PER CENT OF THE TINE.'/)
      GO TO 150
   75 WRITE(5,80)
   80 FORMAT(1H1,'OVERFLOW IN CHI')
  150 CONTINUE
      END
```

```
.A0.TEST2
      DIMENSION MA(16,16)
      READ(8,5) NT
    5 FORMAT(I5)
      WRITE(5,32)
   32 FORMAT(1H1,'OUTPUT FROM IRAND'//)
      WRITE(5,7) NT
    7 FORMAT(1X,12X,'TEST2 (SECOND TEST OF IRAND) TESTS WHETHER
     1 ADJACENT PAIRS OF'/13X,'NUMBERS GENERATED BY RAND BEHAVE AS
     2 THOUGH DRAWN FROM A'/13X,'UNIFORMLY DISTRIBUTED POPULATION
     3 OF PAIRS'/13X,'RESULTS OBTAINED FROM',I4,'EXECUTIONS OF THIS
     4 TEST ARE GIVEN BELOW.')
      DO 90IT=1,NT
C  READ THE INPUT VALUES.**NP** DENOTES THE NUMBER OF PAIRS.
C **NG** DENOTES THE NUMBER OF GROUPS.
      READ (8,10) NP,NG ,MS
   10 FORMAT(2I5,I12)
      DO 15I=1,NG
      DO 15J=1,NG
   15 MA(I,J)=0
C **NF** DENOTES THE DEGREES OF FREEDOM.
      NF=NG**2-1
      MSTART=MS
      CALL PINAX(MS)
      NB=2**34
      NB=NB/NG
      DO 20 K=1,NP
      L=IRAND(MS)
      I=(L/(2*NB))+1
      LA=IRAND(MS)
      J=(LA/(2*NB))+1
   20 MA(I,J)=MA(I,J)+1
C CALCULATE THE CHI SQUARE STATISTIC.
      K=0
      DO 30I=1,NG
      DO 30J=1,NG
   30 K=K+MA(I,J)**2
      V=K*NG*NG
      U=NP
      V=(V/U)-U
      VAR=CHI(V,NF,$75)*100.
     VAR=100.-VAR
      VAR=100.-VAR
      NGSQ=NG**2
      WRITE(5,40) IT,NP,NGSQ,MSTART,V,NF,VAR
   40 FORMAT(12X,'TEST',2X,I3,3X,I5,2X,'PAIRS OF NUMBERS IN'I4,3X,'GROU
     1 PS'//12X,'STARTING NUMBER=',1X,I12//13X,'CHI SQUARE STATISTIC=',F
     213.3,2X,'DEGREE OF FREEDOM=',I5,')'/3X,'STATISTIC SHOULD BE GREATE
     3 R THAN THE ABOVE VALUE',1X,F8.2,'OF TIME.'//)
      GO TO 90
   75 WRITE (5,80)
   80 FORMAT (1H1,12X,'OVERFLOW OCCURED IN CHI')
   90 CONTINUE
      END
```

CHI SQUARE STATISTIC= 10.720 DEGREE OF FREEDOM= 15)
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 22.780F TIME. (continued)

TEST 6 200 PAIRS OF NUMBERS IN 16 GROU PS

STARTING NUMBER= 9

CHI SQUARE STATISTIC= 8.640 DEGREE OF FREEDOM= 15)
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 10.440F TIME.

TEST 7 200 PAIRS OF NUMBERS IN 16 GROU PS

STARTING NUMBER= 3141592

CHI SQUARE STATISTIC= 21.120 DEGREE OF FREEDOM= 15)
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 86.700F TIME.

TEST 8 200 PAIRS OF NUMBERS IN 16 GROU PS

STARTING NUMBER= 122070314

CHI SQUARE STATISTIC= 9.920 DEGREE OF FREEDOM= 15)
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 17.530F TIME.

TEST 9 2000 PAIRS OF NUMBERS IN 100 GROU PS

STARTING NUMBER= 117

CHI SQUARE STATISTIC= 106.900 DEGREE OF FREEDOM= 99)
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 72.380F TIME.

Continued

TEST 10 2000 PAIRS OF NUMBERS IN 100 GROU PS

STARTING NUMBER= 131

CHI SQUARE STATISTIC= 104.500 DEGREE OF FREEDOM= 99)
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 66.680F TIME.

TEST 11 2000 PAIRS OF NUMBERS IN 100 GROU PS

STARTING NUMBER= 8924681

CHI SQUARE STATISTIC= 98.100 DEGREE OF FREEDOM= 99)
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 49.330F TIME.

TEST 12 2000 PAIRS OF NUMBERS IN 100 GROU PS

STARTING NUMBER= 1591917152

CHI SQUARE STATISTIC= 98.700 DEGREE OF FREEDOM= 99)
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 51.040F TIME.

OUTPUT FROM HRAND

THARN2 (SECOND TEST OF RAND) TESTS WHETHER ADJACENT PAIRS OF NUMBERS GENERATED BY RAND BEHAVE AS THOUGH DRAWN FROM A UNIFORMLY DISTRIBUTED POPULATION OF PAIRS
RESULTS OBTAINED FROM 12EXECUTIONS OF THIS TEST ARE GIVEN BELOW,

TEST 1 60 PAIRS OF NUMBERS IN 4 GROUPS

STARTING NUMBER= 19

CHI SQUARE STATISTIC= 3.333 DEGREE OF FREEDOM= 3)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE 65.70% OF TIME.

TEST 2 60 PAIRS OF NUMBERS IN 4 GROUPS

STARTING NUMBER= 23

CHI SQUARE STATISTIC= .800, DEGREE OF FREEDOM= 3)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE 15.05% OF TIME.

TEST 3 60 PAIRS OF NUMBERS IN 4 GROUPS

STARTING NUMBER= 12345

CHI SQUARE STATISTIC= 4.933 DEGREE OF FREEDOM= 3)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE 82.33% OF TIME.

TEST 4 60 PAIRS OF NUMBERS IN 4 GROUPS

STARTING NUMBER= 543212

CHI SQUARE STATISTIC= .667 DEGREE OF FREEDOM= 3)
STATISTIC SHOULD BE GREATER THAN THE ABOVE VALUE 11.90% OF TIME.

TEST 5 200 PAIRS OF NUMBERS IN 16 GROUPS

STARTING NUMBER= 5

CHI SQUARE STATISTIC= 13.760 DEGREE OF FREEDOM= 15)
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 45.620F TIME.

(continued)

TEST 6 200 PAIRS OF NUMBERS IN 16 GROU PS

STARTING NUMBER= 9

CHI SQUARE STATISTIC= 16.160 DEGREE OF FREEDOM= 15)
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 62.850F TIME.

TEST 7 200 PAIRS OF NUMBERS IN 16 GROU PS

STARTING NUMBER= 3141592

CHI SQUARE STATISTIC= 20.320 DEGREE OF FREEDOM= 15)
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 84.000F TIME.

TEST 8 200 PAIRS OF NUMBERS IN 16 GROU PS

STARTING NUMBER= 122070314

CHI SQUARE STATISTIC= 19.520 DEGREE OF FREEDOM= 15)
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 80.890F TIME.

TEST 9 2000 PAIRS OF NUMBERS IN 100 GROU PS

STARTING NUMBER= 117

(continued)

CHI SQUARE STATISTIC= 86.500 DEGREE OF FREEDOM= 991
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 18.920F TIME.

TEST 10 2000 PAIRS OF NUMBERS IN 100 GROU PS

STARTING NUMBER= 131

CHI SQUARE STATISTIC= 67.300 DEGREE OF FREEDOM= 991
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE .620F TIME.

TEST 11 2000 PAIRS OF NUMBERS IN 100 GROU PS

STARTING NUMBER= 8924681

CHI SQUARE STATISTIC= 109.300 DEGREE OF FREEDOM= 991
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 77.490F TIME.

TEST 12 2000 PAIRS OF NUMBERS IN 100 GROU PS

STARTING NUMBER= 1591917152

CHI SQUARE STATISTIC= 104.100 DEGREE OF FREEDOM= 991
STATISTIC SHOULD BE GREATE R THAN THE ABOVE VALUE 65.670F TIME.

```
GOLF0:[PAO.TEST4
      1             DIMENSION NX(15)
      2       C RUNS TEST OF PSEUDO-RANDOM GENERATOR IRAND. IN EACH TEST MANY
      3       C EQUAL LENGTHED SEQUENCES OF NUMBERS WERE GENERATED, THE NUMBER OF
      4       C LARGE RUNS (UP AND DOWN) IN EACH GROUP WAS COMPUTED, AND THE
      5       C TOTAL FOR ALL GROUPS WAS COMPARED WITH THE TOTAL ONE WOULD
      6       C EXPECT IF THE NUMBERS WERE TRULY RANDOM.
      7             WRITE(5,13)
      8        13   FORMAT(1H1,'OUTPUT FROM IRAND'//)
      9             READ(8,7) N
     10        7    FORMAT(I5)
     11             WRITE(5,8)
     12       8     FORMAT(8X,'EXP NUM OF RUNS',2X,'ACTUAL NUM OF RUNS',2X,'PROB'/
     13             DO 200 NTEST=1,N
     14             READ(8,10) NG,LG,MS
     15        10   FORMAT(2I5,I12)
     16             CALL PINAX(MS)
     17             MIN=(LG+1)/2
     18             NCOUNT=0
     19             DO 100 I=1,NG
     20       C GENERATION OF A GROUP OF LG CONSECUTIVE NUMBERS.
     21             DO 15 J=1,LG
     22       15    NX(J)=IRAND(MS)
     23             NU=0
     24             ND=0
     25             J=0
     26        20   J=J+1
     27             IF(NX(J).LT.NX(J+1)) GO TO 25
     28       C INCREMENT OF RUNS DOWN BY ONE.
     29             ND=ND+1
     30             IF(NU.EQ.0) GO TO 30
     31        22   IF(NU.GE.MIN) GO TO 40
     32             NU=0
     33        24   IF(J.LT.(LG-1)) GO TO 20
     34             GO TO 100
     35       C INCREMENT OF RUNS UP BY ONE.
     36         25 NU=NU+1
     37             IF(ND.EQ.0) GO TO 35
     38        28   IF(ND.GE.MIN) GO TO 40
     39             ND=0
     40             GO TO 24
     41        30   IF(J.LT.(LG-1)) GO TO 20
     42             GO TO 28
     43        35   IF(J.LT.(LG-1)) GO TO 20
     44             GO TO 22
     45       C INCREMENT OF NCOUNT BY ONE. NCOUNT CONTAINS THE TOTAL NUMBER
     46       C OF RUNS GREATER THAN OR EQUAL TO LG/2.
     47        40   NCOUNT=NCOUNT+1
     48        100  CONTINUE
     49       C COMPUTATION OF THE PROBABILITY P=2*(LG*(MIN+1)-(MIN**2+MIN-1))
     50       C DEVIDED BY (MIN+2)!
     51             U=NCOUNT
     52             V=NG
     53             J=1
     54             K=MIN+2
     55       C CALCULATION OF (MIN+2)!
     56             DO 110 I=1,K
```

(continued)

```
      110   J=J*I
            U=2*(LG*(MIN+1)-(MIN**2+MIN-1))
            V=J
            P=U/V
      C ** PL ** IS THE PROBABILITY OF OBTAINING RUNS EQUAL OR LESS TO NCOUNT.
            PL=BIN(NCOUNT,NG,P)*100.
      C **V** DENOTES THE EXPECTED NUMBER OF RUNS
            V=P*Q
            WRITE(5,135) V,NCOUNT,PL
      135   FORMAT(12X,F6.1,11X,I5,9X,F8.4)
      200   CONTINUE
            END


AO.NEIN1,.PASBIG1,.IPRIME1
```

OUTPUT FROM IRAND

EXP NUM OF RUNS	ACTUAL NUM OF RUNS	PROB
166.7	155	14.4717
166.7	150	6.2551
166.7	168	56.9038
166.7	163	38.1930
166.7	167	53.1506
166.7	171	67.6714
166.7	158	21.9241
166.7	159	24.8287
166.7	170	64.1945
166.7	156	16.7399
166.7	162	34.6316
166.7	147	3.4508
166.7	160	27.9267
150.0	161	84.5770
150.0	157	74.6722
150.0	173	98.1292
150.0	139	17.6213
150.0	150	51.7660
150.0	154	65.4879
150.0	139	17.6213
150.0	154	65.4879
150.0	156	71.7574
150.0	146	37.8294
150.0	147	41.2389
150.0	145	34.5121
150.0	139	17.6213
36.9	31	18.5338
36.9	37	53.9268
36.9	36	47.3274
36.9	24	1.9957
36.9	31	18.5338
36.9	43	86.2669
36.9	28	8.1945
36.9	52	99.5104
36.9	46	94.4002
36.9	42	82.2973
36.9	40	72.4242
36.9	43	86.2669
36.9	34	34.5202

OUTPUT FROM GRAND

EXP NUM OF RUNS	ACTUAL NUM OF RUNS	PROB
166.7	151	7.5098
166.7	171	67.6714
166.7	167	53.1506
166.7	175	79.8986
166.7	168	56.9038
166.7	162	34.6316
166.7	157	19.2252
166.7	168	56.9038
166.7	180	90.5298
166.7	164	41.8572
166.7	164	41.8572
166.7	173	74.1594
166.7	162	34.6316
150.0	142	25.3278
150.0	144	31.3098
150.0	152	58.7611
150.0	164	90.0454
150.0	148	44.7159
150.0	141	22.5793
150.0	141	22.5793
150.0	151	55.2841
150.0	152	58.7611
150.0	157	74.6722
150.0	148	44.7159
150.0	141	22.5793
150.0	161	84.5770
36.9	47	96.0363
36.9	34	34.5202
36.9	32	23.2826
36.9	27	5.9648
36.9	37	53.9268
36.9	41	77.6708
36.9	27	5.9648
36.9	38	60.4196
36.9	40	72.4242
36.9	33	60.4196
36.9	36	47.3274
36.9	37	53.9268
36.9	31	18.5338

OLEO*DAO.DATA4

1	39		
2	500	3	37
3	500	3	38
4	500	3	39
5	500	3	40
6	500	3	41
7	500	3	3770
8	500	3	1234321
9	500	3	777666555
10	500	3	9876543212
11	500	3	10000000000
12	500	3	10000000001
13	500	3	10000000002
14	500	3	10000000003
15	1000	5	11
16	1000	5	12
17	1000	5	13
18	1000	5	14
19	1000	5	15
20	1000	5	3643
21	1000	5	15273
22	1000	5	564737
23	1000	5	2510733
24	1000	5	11000000000
25	1000	5	11000000001
26	1000	5	11000000002
27	1000	5	11000000003
28	3000	10	17
29	3000	10	18
30	3000	10	19
31	3000	10	20
32	3000	10	21
33	3000	10	3374
34	3000	10	104526
35	3000	10	16146334
36	3000	10	1155562176
37	3000	10	11100000000
38	3000	10	11100000001
39	3000	10	11100000002
40	3000	10	11100000003

```
FORTRAN,TEST
      C GAP TEST.
      C THIS TEST EXAMINES THE LENGTH OF GAPS BETWEEN OCCURRENCES OFU(J)
      C IN A CERTAIN RANGE.
            INTEGER T,S,D,R,RI,COUNT
            DIMENSION COUNT(20),V(20),P(20),RI(20)
            READ(3,1500) NN
      1500  FORMAT(I5)
            WRITE(5,40)
      40    FORMAT(1H1,'OUTPUT FROM IRAND'/)
            DO 150 NTEST=1,NN
            READ(3,2000) A,B,T,N,MS
      2000  FORMAT(2F3.2,I2,I4,I12)
            WRITE(5,999) NTEST,N,A,B
       999  FORMAT(/10X,'TEST NO',1X,I2,2X,'TOTAL NUMBER OF GAPS',1X,I4,1X,'
           1RANGE IN WHICH ARE EXAMINED',1X,F3.2,3X,F3.2)
            SUM=0.
            CALL PINAX(MS)
      C INITIALIZE.
            S=0
            D=T+1
            R=1
        3   COUNT(R)=0
            R=R+1
            IF(R.GT.D) GO TO 4
            GO TO 3
        4   CONTINUE
      C SET R EQUAL TO ONE.
       10   R=1
      C GENERATION OF A RANDOM NUMBER IN THE RANGE (0,2**35-1).
        5   IZ=IRAND(MS)
            U=IZ/2.**35
            IF(U.GE.A.AND.U.LT.B) GO TO 6
            R=R+1
            GO TO 5
        6   IF(R.GE.D) GO TO 8
      C RECORD A GAP LENGTH.
            COUNT(R)=COUNT(R)+1
            GO TO 7
        8   COUNT(D)=COUNT(D)+1
        7   S=S+1
      C CHECK IF N GAPS HAVE BEEN FOUND.
            IF(S.LT.N) GO TO 10
            DO 51 I=1,T
       51   RI(I)=I-1
      C CALCULATION OF THE QUI-SQUARE STATISTIC.
            NF=D-1
            DO 41 I=1,D
       41   V(I)=COUNT(I)
            DO 42 I=1,D
       42   V(I)=V(I)**2
            P1=B-A
            PP=1-P1
           P(1)=P1
            I=2
       43   K=I-1
            P(I)=P1*PP**K
```

(continued)

```
57              I=I+1
58              IF(I.EQ.D) GO TO 44
59              GO TO 43
60        44    K=D-I
61              P(D)=PP**K
62              DO 50 I=1,D
63              V(I)=V(I)/P(I)
64              SUM=SUM+V(I)
65        50    CONTINUE
66              SUM=(1./FLOAT(N))*SUM
67              VV=SUM-FLOAT(N)
68              VAR=CHI(VV,NF,$75)
69              VAR=VAR*100.
70              VAR=100.-VAR
71              WRITE(5,1000) VV,VAR
72        1000  FORMAT(10X,'CHI-SQUARE STATISTIC',1X,F8.3,1X,'PROBAB.',1X,F8.3/)
73              GO TO 150
74        75    WRITE(5,80)
75        80    FORMAT(1H1,'OVER OCCURED IN FUNCTION CHI')
76        150   CONTINUE
77              END
```

OUTPUT FROM IRAND

TEST NO 1	TOTAL NUMBER OF GAPS 256	RANGE IN WHICH ARE EXAMINED	.00	.10
CHI-SQUARE STATISTIC	.571 PROBAB.	96.626		
TEST NO 2	TOTAL NUMBER OF GAPS 256	RANGE IN WHICH ARE EXAMINED	.00	.10
CHI-SQUARE STATISTIC	3.502 PROBAB.	38.599		
TEST NO 3	TOTAL NUMBER OF GAPS 512	RANGE IN WHICH ARE EXAMINED	.00	.10
CHI-SQUARE STATISTIC	4.258 PROBAB.	37.223		
TEST NO 4	TOTAL NUMBER OF GAPS 512	RANGE IN WHICH ARE EXAMINED	.00	.10
CHI-SQUARE STATISTIC	2.893 PROBAB.	27.264		
TEST NO 5	TOTAL NUMBER OF GAPS 1024	RANGE IN WHICH ARE EXAMINED	.00	.10
CHI-SQUARE STATISTIC	8.444 PROBAB.	7.660		
TEST NO 6	TOTAL NUMBER OF GAPS 1024	RANGE IN WHICH ARE EXAMINED	.00	.10
CHI-SQUARE STATISTIC	4.817 PROBAB.	77.693		
TEST NO 7	TOTAL NUMBER OF GAPS 2048	RANGE IN WHICH ARE EXAMINED	.00	.50
CHI-SQUARE STATISTIC	13.723 PROBAB.	8.929		
TEST NO 8	TOTAL NUMBER OF GAPS 2048	RANGE IN WHICH ARE EXAMINED	.00	.50
CHI-SQUARE STATISTIC	.952 PROBAB.	91.696		
TEST NO 9	TOTAL NUMBER OF GAPS 1000	RANGE IN WHICH ARE EXAMINED	.50	.70
CHI-SQUARE STATISTIC	15.785 PROBAB.	10.597		

CONTINUED

TEST NO 10 TOTAL NUMBER OF GAPS 500 RANGE IN WHICH ARE EXAMINED .40 .45
CHI-SQUARE STATISTIC 4.428 PROBAB. 88.954

TEST NO 11 TOTAL NUMBER OF GAPS 512 RANGE IN WHICH ARE EXAMINED .30 .38
CHI-SQUARE STATISTIC 9.461 PROBAB. 48.901

TEST NO 12 TOTAL NUMBER OF GAPS 1280 RANGE IN WHICH ARE EXAMINED .50 .75
CHI-SQUARE STATISTIC 4.754 PROBAB. 96.570

TEST NO 13 TOTAL NUMBER OF GAPS 1024 RANGE IN WHICH ARE EXAMINED .12 .33
CHI-SQUARE STATISTIC 9.803 PROBAB. 27.914

TEST NO 14 TOTAL NUMBER OF GAPS 2048 RANGE IN WHICH ARE EXAMINED .45 .46
CHI-SQUARE STATISTIC 13.454 PROBAB. 33.693

TEST NO 15 TOTAL NUMBER OF GAPS 4096 RANGE IN WHICH ARE EXAMINED .95 .96
CHI-SQUARE STATISTIC 10.742 PROBAB. 21.673

TEST NO 16 TOTAL NUMBER OF GAPS 128 RANGE IN WHICH ARE EXAMINED .01 .20
CHI-SQUARE STATISTIC 2.008 PROBAB. 73.426

TEST NO 17 TOTAL NUMBER OF GAPS 2048 RANGE IN WHICH ARE EXAMINED .35 .36
CHI-SQUARE STATISTIC 2.071 PROBAB. 97.877

TEST NO 18 TOTAL NUMBER OF GAPS 4096 RANGE IN WHICH ARE EXAMINED .01 .12
CHI-SQUARE STATISTIC 6.831 PROBAB. 97.639

TEST NO 19 TOTAL NUMBER OF GAPS 500 RANGE IN WHICH ARE EXAMINED .80 .90
CHI-SQUARE STATISTIC 3.919 PROBAB. 41.703

TEST NO 20 TOTAL NUMBER OF GAPS 1000 RANGE IN WHICH ARE EXAMINED .72 .73
CHI-SQUARE STATISTIC 3.211 PROBAB. 92.042

GOLFO*PAO.DATAS

```
      1           20
      2       00001004025600000000001
      3       00001008025600000000001438
      4       00001004051200008536741 9
      5       00001008051 2 300025896 74
      6       00001004102 4    789520146
      7       00001008102 4   1235860120
      8       00005008204 8   7889654230
      9       00005004204 8   7773564203
     10       05007010100 0      5423879
     11       04004512050 0   3560147853
     12       03003810051200111111112
     13       05007512128000000000123
     14       01203308192400012345678 9
     15       04504612204800011134511 1
     16       09509608409600033333333 4
     17       00102004012800000000002 5
     18       03503606204800000000004 2
     19       00101216409600123456789 1
     20       08009004050000000000012 5
     21       07207308100000045789189 9
```

TEST NO 1 TOTAL NUMBER OF GAPS 256 RANGE IN WHICH ARE EXAMINED .00 .10
CHI-SQUARE STATISTIC 3.540 PROBAB. 47.180

TEST NO 2 TOTAL NUMBER OF GAPS 256 RANGE IN WHICH ARE EXAMINED .00 .10
CHI-SQUARE STATISTIC 5.177 PROBAB. 73.848

TEST NO 3 TOTAL NUMBER OF GAPS 512 RANGE IN WHICH ARE EXAMINED .00 .10
CHI-SQUARE STATISTIC 4.542 PROBAB. 33.760

TEST NO 4 TOTAL NUMBER OF GAPS 512 RANGE IN WHICH ARE EXAMINED .00 .10
CHI-SQUARE STATISTIC 5.596 PROBAB. 37.752

TEST NO 5 TOTAL NUMBER OF GAPS 1024 RANGE IN WHICH ARE EXAMINED .00 .10
CHI-SQUARE STATISTIC 9.338 PROBAB. 5.318

TEST NO 6 TOTAL NUMBER OF GAPS 1024 RANGE IN WHICH ARE EXAMINED .00 .10
CHI-SQUARE STATISTIC 8.303 PROBAB. 31.741

TEST NO 7 TOTAL NUMBER OF GAPS 2048 RANGE IN WHICH ARE EXAMINED .00 .50
CHI-SQUARE STATISTIC 8.650 PROBAB. 37.264

TEST NO 8 TOTAL NUMBER OF GAPS 2048 RANGE IN WHICH ARE EXAMINED .00 .50
CHI-SQUARE STATISTIC 6.767 PROBAB. 14.875

TEST NO 9 TOTAL NUMBER OF GAPS 1000 RANGE IN WHICH ARE EXAMINED .50 .70
CHI-SQUARE STATISTIC 5.314 PROBAB. 86.925

(CONTINUED)

TEST NO 10 TOTAL NUMBER OF GAPS 500 RANGE IN WHICH ARE EXAMINED .40 .45
CHI-SQUARE STATISTIC 15.490 PROBAB. 21.823

TEST NO 11 TOTAL NUMBER OF GAPS 512 RANGE IN WHICH ARE EXAMINED .30 .38
CHI-SQUARE STATISTIC 13.187 PROBAB. 21.340

TEST NO 12 TOTAL NUMBER OF GAPS 1280 RANGE IN WHICH ARE EXAMINED .50 .75
CHI-SQUARE STATISTIC 16.804 PROBAB. 54.582

TEST NO 13 TOTAL NUMBER OF GAPS 1024 RANGE IN WHICH ARE EXAMINED .12 .33
CHI-SQUARE STATISTIC 7.317 PROBAB. 50.287

TEST NO 14 TOTAL NUMBER OF GAPS 2048 RANGE IN WHICH ARE EXAMINED .45 .46
CHI-SQUARE STATISTIC 13.373 PROBAB. 34.249

TEST NO 15 TOTAL NUMBER OF GAPS 4096 RANGE IN WHICH ARE EXAMINED .95 .96
CHI-SQUARE STATISTIC 7.829 PROBAB. 45.036

TEST NO 16 TOTAL NUMBER OF GAPS 128 RANGE IN WHICH ARE EXAMINED .01 .20
CHI-SQUARE STATISTIC 15.780 PROBAB. .333

TEST NO 17 TOTAL NUMBER OF GAPS 2048 RANGE IN WHICH ARE EXAMINED .35 .36
CHI-SQUARE STATISTIC 3.109 PROBAB. 92.735

TEST NO 18 TOTAL NUMBER OF GAPS 4096 RANGE IN WHICH ARE EXAMINED .01 .12
CHI-SQUARE STATISTIC 20.270 PROBAB. 20.832

TEST NO 19 TOTAL NUMBER OF GAPS 500 RANGE IN WHICH ARE EXAMINED .80 .90
CHI-SQUARE STATISTIC 2.759 PROBAB. 59.369

TEST NO 20 TOTAL NUMBER OF GAPS 1000 RANGE IN WHICH ARE EXAMINED .72 .73
CHI-SQUARE STATISTIC 9.355 PROBAB. 31.322

```
RUN,LIST.
      C POKER TEST.
            INTEGER S,DIF,D,OCCURS(30),CATEG(10),R,T
            READ(3,18) IN
      18    FORMAT(I5)
            WRITE(5,998)
      998   FORMAT(1H1,'OUTPUT FROM IRAND'///)
            DO 1000 NTEST=1,IN
            READ(3,10) NG,D,NS,T
      10    FORMAT(I6,I4,I12,I2)
            IVAL=NS
            CALL PINAX(NS)
            SUM=0.
            V=0.
            DEXP1=D**T
            DO 1 I=1,T
      1     CATEG(I)=0
            S=0
      6     DO 2 I=1,D
      2     OCCURS(I)=0
            DIF=0
            K=1
      4     U=IRAND(NS)/(2.**35)
      C THE RANDOM NUMBER U IS CONVERTED INTO AN INTEGER IN THE RANGE 1,D
            L=D*U+1
            IF(OCCURS(L).NE.0) GO TO 3
            OCCURS(L)=1
            DIF=DIF+1
      3     K=K+1
            IF(K.GT.T) GO TO 5
            GO TO 4
      5     CATEG(DIF)=CATEG(DIF)+1
            S=S+1
            IF(S.LT.NG) GO TO 6
      C THE LUMP TWO CATEGORIES TOGETHER.
            CATEG(1)=CATEG(1)+CATEG(2)
            M1=T-1
            DO 20 I=2,M1
      20    CATEG(I)=CATEG(I+1)
      C V IS THE PROBABILITY A NUMBER TO BELONG IN TO CATEG(1) OR CATEG(2).
            DO 21 R=1,2
      21    V=V+(KORDER(D,R)/DEXP1)*ISTIR(T,R)
      C CALCULATION OF THE CHI-SQUARE STATISTIC.
            CATEG(1)=CATEG(1)**2
            CATEG(1)=CATEG(1)/V
            DO 22 I=2,M1
            CATEG(I)=CATEG(I)**2
            V1=(KORDER(D,I+1)/DEXP1)*ISTIR(T,I+1)
      22    CATEG(I)=CATEG(I)/V1
            DO 23 I=1,M1
      23    SUM=SUM+CATEG(I)
            SUM=SUM/NG
            V1=SUM-NG
            M1=M1-1
            VAR=CHI(V1,M1,$75)*100.
            VAR=100.-VAR
            WRITE(5,501) IVAL,M1,V1,VAR
      501   FORMAT(10X,'STARTING NUMBER ',I8,//,
           1X,'CHI-SQUARE STATISTIC',1X,F7.3,1X,'STATISTIC SHOULD BE GREATER
           2THAN THIS VALUE',1X,F8.2,'PERCENT OF THE TIME'//)
            GO TO 1000                                    1000 CONTINUE
      75    WRITE(5,999)
      999   FORMAT(1X,'OVERFLOW IN FUNCTION CHI')
```

OUTPUT FROM IRAND

STARTING NUMBER	1	DEGREES OF FREEDOM	3		
CHI-SQUARE STATISTIC	1.510	STATISTIC SHOULD BE GREATER THAN THIS VALUE		68.00	PERCENT OF THE TIME
STARTING NUMBER	12398	DEGREES OF FREEDOM	3		
CHI-SQUARE STATISTIC	1.950	STATISTIC SHOULD BE GREATER THAN THIS VALUE		58.28	PERCENT OF THE TIME
STARTING NUMBER	3452890657	DEGREES OF FREEDOM	3		
CHI-SQUARE STATISTIC	1.480	STATISTIC SHOULD BE GREATER THAN THIS VALUE		68.69	PERCENT OF THE TIME
STARTING NUMBER	1275054768	DEGREES OF FREEDOM	3		
CHI-SQUARE STATISTIC	5.090	STATISTIC SHOULD BE GREATER THAN THIS VALUE		16.53	PERCENT OF THE TIME
STARTING NUMBER	44498906	DEGREES OF FREEDOM	4		
CHI-SQUARE STATISTIC	.331	STATISTIC SHOULD BE GREATER THAN THIS VALUE		98.77	PERCENT OF THE TIME
STARTING NUMBER	666666	DEGREES OF FREEDOM	4		
CHI-SQUARE STATISTIC	7.679	STATISTIC SHOULD BE GREATER THAN THIS VALUE		10.41	PERCENT OF THE TIME
STARTING NUMBER	554987	DEGREES OF FREEDOM	4		
CHI-SQUARE STATISTIC	4.301	STATISTIC SHOULD BE GREATER THAN THIS VALUE		36.66	PERCENT OF THE TIME
STARTING NUMBER	44388899	DEGREES OF FREEDOM	4		
CHI-SQUARE STATISTIC	.709	STATISTIC SHOULD BE GREATER THAN THIS VALUE		95.02	PERCENT OF THE TIME
STARTING NUMBER	553376	DEGREES OF FREEDOM	5		
CHI-SQUARE STATISTIC	6.433	STATISTIC SHOULD BE GREATER THAN THIS VALUE		26.63	PERCENT OF THE TIME
STARTING NUMBER	432767	DEGREES OF FREEDOM	5		
CHI-SQUARE STATISTIC	9.343	STATISTIC SHOULD BE GREATER THAN THIS VALUE		9.62	PERCENT OF THE TIME

CONTINUED

STARTING NUMBER 321232 DEGREES OF FREEDOM 5
CHI-SQUARE STATISTIC 6.330 STATISTIC SHOULD BE GREATER THAN THIS VALUE 27.54 PERCENT OF THE TIME

STARTING NUMBER 200200000 DEGREES OF FREEDOM 5
CHI-SQUARE STATISTIC 17.446 STATISTIC SHOULD BE GREATER THAN THIS VALUE .37 PERCENT OF THE TIME

STARTING NUMBER 2 DEGREES OF FREEDOM 6
CHI-SQUARE STATISTIC 12.364 STATISTIC SHOULD BE GREATER THAN THIS VALUE 5.43 PERCENT OF THE TIME

STARTING NUMBER 33 DEGREES OF FREEDOM 6

CHI-SQUARE STATISTIC 4.271 STATISTIC SHOULD BE GREATER THAN THIS VALUE 64.01 PERCENT OF THE TIME

STARTING NUMBER 444 DEGREES OF FREEDOM 6
CHI-SQUARE STATISTIC 6.770 STATISTIC SHOULD BE GREATER THAN THIS VALUE 34.18 PERCENT OF THE TIME

STARTING NUMBER 200847 DEGREES OF FREEDOM 6
CHI-SQUARE STATISTIC 1.700 STATISTIC SHOULD BE GREATER THAN THIS VALUE 94.51 PERCENT OF THE TIME

OUTPUT FROM MRAND

STARTING NUMBER	1	DEGREES OF FREEDOM	3		
CHI-SQUARE STATISTIC	9.777	STATISTIC SHOULD BE GREATER THAN THIS VALUE		2.06	PERCENT OF THE TIME
STARTING NUMBER	12398	DEGREES OF FREEDOM	3		
CHI-SQUARE STATISTIC	.283	STATISTIC SHOULD BE GREATER THAN THIS VALUE		96.31	PERCENT OF THE TIME
STARTING NUMBER	3452890657	DEGREES OF FREEDOM	3		
CHI-SQUARE STATISTIC	7.510	STATISTIC SHOULD BE GREATER THAN THIS VALUE		5.73	PERCENT OF THE TIME
STARTING NUMBER	1276054768	DEGREES OF FREEDOM	3		
CHI-SQUARE STATISTIC	1.193	STATISTIC SHOULD BE GREATER THAN THIS VALUE		75.46	PERCENT OF THE TIME
STARTING NUMBER	44409906	DEGREES OF FREEDOM	4		
CHI-SQUARE STATISTIC	4.711	STATISTIC SHOULD BE GREATER THAN THIS VALUE		31.83	PERCENT OF THE TIME
STARTING NUMBER	666666	DEGREES OF FREEDOM	4		
CHI-SQUARE STATISTIC	1.318	STATISTIC SHOULD BE GREATER THAN THIS VALUE		85.83	PERCENT OF THE TIME
STARTING NUMBER	554987	DEGREES OF FREEDOM	4		
CHI-SQUARE STATISTIC	5.431	STATISTIC SHOULD BE GREATER THAN THIS VALUE		24.59	PERCENT OF THE TIME
STARTING NUMBER	44388899	DEGREES OF FREEDOM	4		
CHI-SQUARE STATISTIC	4.778	STATISTIC SHOULD BE GREATER THAN THIS VALUE		31.08	PERCENT OF THE TIME
STARTING NUMBER	553376	DEGREES OF FREEDOM	5		
CHI-SQUARE STATISTIC	.691	STATISTIC SHOULD BE GREATER THAN THIS VALUE		97.09	PERCENT OF THE TIME
STARTING NUMBER	439767	DEGREES OF FREEDOM	5		
CHI-SQUARE STATISTIC	7.011	STATISTIC SHOULD BE GREATER THAN THIS VALUE		21.98	PERCENT OF THE TIME

STARTING NUMBER 321232 DEGREES OF FREEDOM 5
CHI-SQUARE STATISTIC 8.330 STATISTIC SHOULD BE GREATER THAN THIS VALUE 13.90 PERCENT OF THE TIME

STARTING NUMBER 2009000000 DEGREES OF FREEDOM 5
CHI-SQUARE STATISTIC 1.562 STATISTIC SHOULD BE GREATER THAN THIS VALUE 90.58 PERCENT OF THE TIME

STARTING NUMBER 2 DEGREES OF FREEDOM 6
CHI-SQUARE STATISTIC 1.350 STATISTIC SHOULD BE GREATER THAN THIS VALUE 96.88 PERCENT OF THE TIME

STARTING NUMBER 33 DEGREES OF FREEDOM 6

CHI-SQUARE STATISTIC 4.706 STATISTIC SHOULD BE GREATER THAN THIS VALUE 58.20 PERCENT OF THE TIME

STARTING NUMBER 444 DEGREES OF FREEDOM 6
CHI-SQUARE STATISTIC 6.850 STATISTIC SHOULD BE GREATER THAN THIS VALUE 33.41 PERCENT OF THE TIME

STARTING NUMBER 288847 DEGREES OF FREEDOM 6
CHI-SQUARE STATISTIC 7.675 STATISTIC SHOULD BE GREATER THAN THIS VALUE 26.29 PERCENT OF THE TIME

F0*FAO.EXPR1

```
      C SUBROUTINE EXPR GENERATES A RANDOM SEQUENCE OF EXPONENTIALLY
      C DISTRIBUTED NUMBERS.
      C*************************************************************
      C X IS AN ARRAY OF EXPONENTIALLY DISTRIBUTED VARIATES. (OUTPUT)
      C EX IS THE DESIRED MEAN OF THE NUMBERS.
      C N IS THE SIZE OF THE ARRAY X.
      C MS IS THE STARTING VALUE (INTEGER) OF THE SUBROUTINE PINAX.
      C MS SHOULD BE CHOSEN IN THE INTERVAL (0,2**35-1)
      C DIFFERENT VALUES OF MS RESULT IN DIFFERENT OUTPUT SEQUENCES.
      C AT THE END OF THE COMPUTATION MS CONTAINS THE LAST NUMBER
      C GENERATED BY IRAND(INPUT,OUTPUT)
            SUBROUTINE EXPR(X,EX,N,MS)
             DIMENSION X(N)
             CALL PINAX(MS)
            ENTRY EXPO(X,EX,N,MS)
             DO 1 I=1,N
             R=IRAND(MS)/(2.**35)
      1      X(I)=-EX*ALOG(R)
             RETURN
             END
```

F0*FAO.KORDER1

```
      C THIS FUNCTION FINDS THE NUMBER OF ORDERED CHOICES OF L THINGS
      C TAKEN FROM A SET OF N OBJECTS.
      C*************************************************************
      C N IS A POSITIVE INTEGER (INPUT)
      C L IS A POSITIVE INTEGER (INPUT)
      C THE RETURNED VALUE EQUALS TO N(N-1).....(N-L+1)
            FUNCTION KORDER(N,L)
            KORDER=N
            IF(L.NE.1) GO TO 1
            RETURN
      1     K=L
            M=N
      2     K=K-1
            N=N-1
            KORDER=KORDER*N
            IF(K.EQ.1) GO TO 3
            GO TO 2
      3     CONTINUE
            RETURN
            END
```

```
).SORTINTEGER
    C THIS SUBROUTINE SORTS A SET OF REAL NUMBERS IN TO ASCHENDING ORDER.
    C*************************************************************************
    C A IS THE ARRAY OF THE REAL NUMBERS TO BE SORTED.(INPUT AND OUTPUT).
    C N IS THE SIZE OF THE ARRAY A.(INPUT)
          SUBROUTINE SORTRL(N,A)
          DIMENSION A(N)
          I=1
          K=N-1
     5    J=I+1
     6    IF(A(I).LE.A(J)) GO TO 2
          TEMP=A(I)
          A(I)=A(J)
          A(J)=TEMP
     2    CONTINUE
          IF(J.EQ.N) GO TO 3
          J=J+1
          GO TO 6
     3    IF(I.EQ.K) GO TO 4
          I=I+1
          GO TO 5
     4    CONTINUE
          RETURN
           END

.ISTIR1
    C THIS FUNCTION COMPUTES THE STIRLING NUMBERS OF THE SECOND KIND.
    C*************************************************************************
    C M IS A POSITIVE INTEGER OR ZERO.(INPUT)
    C N IS A POSITIVE INTEGER OR ZERO.(INPUT)
    C THE RETURNED VALUE EQUALS TO (M,N)
          FUNCTION ISTIR(M,N)
          ISUM=0
          IF(M.EQ.0) GO TO 3
          IF(N.EQ.0) GO TO 5
          IF(N.EQ.1) GO TO 6
          IF(M.LT.N) GO TO 7
          IF(M.EQ.N) GO TO 8
          GO TO 9
    3     IF(N.EQ.0) GO TO 4
          ISTIR=0
          RETURN
    4     ISTIR=1
          RETURN
    5     ISTIR=0
          RETURN
    6     ISTIR=1
          RETURN
    7     ISTIR=0
          RETURN
    8     ISTIR=1
          RETURN
    9     K=1
    10    L=ICOMB(N,K)
          L=L*(K**M)
          L=(-1)**K*L
          ISUM=ISUM+L
          K=K+1
          IF(K.GT.N) GO TO 11
          GO TO 10
    11    IDIV=(-1)**N
          IDIV=IDIV*IPROD(N)
          ISTIR=ISUM/IDIV
          RETURN
          END
```

```
*MAIN
      C INPUT OF THE SYSTEM.
            DIMENSION X(9),A(9,9),B(9)
            N=9
            EPS=.000001
            MAXIM=1000
            DO 1 I=1,5
            IPLUS4=I+4
            DO 1 J=IPLUS4,9
    1       A(I,J)=0.
            DO 2 I=1,6
    2       A(I,I+3)=-1.
            DO 3 I=1,7
    3       A(I,I+2)=0.
            DO 4 I=1,8
    4       A(I,I+1)=-1.
            A(3,4)=0.
            A(6,7)=0.
            DO 5 I=1,9
    5       A(I,I)=4
            DO 6 I=2,9
            ILESSI=I-1
            DO 6 J=1,ILESSI
    6       A(I,J)=A(J,I)
            DO 7 I=1,6
    7       B(I)=0.
            DO 8 I=7,9
    8       B(I)=1.
            CALL GAUSS(A,B,N,EPS,MAXIM,X,ITER)
            WRITE(5,10) ITER
   10       FORMAT(20X,'NUMBER OF ITERATIONS PERFORMED',1X,I5///)
            WRITE(5,11)
   11       FORMAT(20X,'SOLUTION OF THE SYSTEM'/)
            DO 45 I=1,N
   45       WRITE(5,12) I,X(I)
   12       FORMAT(26X,'X(',I2,')=',1X,F12.5)
            END
```

```
*PAD.COMBSUB
      C THIS FUNCTION CALCULATES N FACTORIAL WHERE N IS AN INTEGER.
      C*************************************************************
      C N IS AN INTEGER.(INPUT)
      C L=IPROD(N) IS THE VALUE OF N FACTORIAL.
            FUNCTION IPROD(N)
             IF(N.EQ.1.OR.N.EQ.0) GO TO 4
            IPROD=1
            I=1
    1       I=I+1
            IF(I-N) 2,2,3
    2       IPROD=IPROD*I
            GO TO 1
    3       CONTINUE
            RETURN
    4       IPROD=1
            RETURN
            END
```

```
-14:36-*GET(1)
0101:          INTEGER X
0102:          DIMENSION X(13)
0103:          N=13
0104:          DO 6 I=1,N
0105:          X(I)=I
0106:    6     CONTINUE
0107:          MS=555436780
0110:          M=20000
0111:          ISUM1=0
0112:          ISUM2=0
0113:          KOUNT=1
       C SUBROUTINE SHUFFL SHUFFLES THE DECK OF THIRTEEN CARDS
0114:          CALL SHUFFL(X,N,MS)
0115:    4     K=1
0116:          I=1
       C HAS A MEETING BEEN OCCURED
0117:    1     IF(X(I).EQ.K) GO TO 3
0120:          K=K+1
0121:          I=I+1
0122:          IF(I.GT.N) GO TO 2
0123:          GO TO 1
       C INCREASE THE NUMBER OF SUCCESSES OF PLAYER B BY ONE.
0124:    2     ISUM2=ISUM2+1
0125:          GO TO 15
       C INCREASE THE NUMBER OF SUCCESSES OF PLAYER A BY ONE.
0126:    3     ISUM1=ISUM1+1
0127:   15      KOUNT=KOUNT+1
0130:          IF(KOUNT.GT.M) GO TO 5
0131:          CALL PERMUT(X,N,MS)
0132:          GO TO 4
       C CALCULATE THE PROBABILITY PA AND PB.
0133:    5     PA=FLOAT(ISUM1)/FLOAT(M)
0134:          PB=FLOAT(ISUM2)/FLOAT(M)
0135:          WRITE(5,10) PA,PB
0136:   10     FORMAT(10X,'PROBABILITY THE GAME TO BE WON BY A=',F7.4/10X,'PROBA
           1BILITY THE GAME TO BE WON BY B=',F7.4)
0137:          END
```

```
1B1
: THIS FUNCTION FINDS THE COMBINATIONS K THINGS TAKEN FROM N OBJECTS.
: ***********************************************************************
: N IS A POSITIVE INTEGER (INPUT)
: K IS A POSITIVE INTEGER (INPUT)
       FUNCTION ICOMB(N,K)
       ICOMB=IPROD(N)/((IPROD(K))*(IPROD(N-K)))
       RETURN
       END
```

```
.COLLECT
   C COUPON COLLECTORS TEST.
         INTEGER Q,R,S,D1,TMIN1,D,T,T1,COUNT(100),OCCURS(100)
         READ(8,11) NN
    11   FORMAT(I3)
         WRITE(5,502)
    502  FORMAT(1H1,'OUTPUT FROM IRAND'///)
         DO 1000 NTEST=1,NN
         READ(8,10) N,D,T,MS
    10   FORMAT(I6,I4,I5,I12)
         IVAL=MS
         NN=IPROD(D)
         SUM=0
         T1=T+1
         D1=D+1
   C INITIALIZE
         S=0
         DO 1 R=1,T1
    1    COUNT(R)=0
         CALL PINAX(MS)
   C SET Q,R EQUAL TO 1.
   100   Q=1
         R=1
         DO 2 K=1,D
    2    OCCURS(K)=0
    3    R=R+1
   C GENERATE A RANDOM NUMBER U UNIFORMLY DISTRIBUTED BETWEEN 0 AND 1.
         U=IRAND(MS)/(2.**35)
   C THE RANDOM NUMBER U IS CONVERTED INTO AN INTEGER BETWEEN 1 AND D.
         L=D*U+1.
         IF(OCCURS(L).NE.0) GO TO 3
   C IS THE SET COMPLETE
         OCCURS(L)=1
         Q=Q+1
         IF(Q.GT.D) GO TO 4
         GO TO 3
    4    IF(R.GE.T1) GO TO 5
   C RECORD THE LENGTH.
         COUNT(R)=COUNT(R)+1
         GO TO 6
    5    COUNT(T1)=COUNT(T1)+1
    6    S=S+1
   C HAVE N LENGTHS FOUND.
         IF(S.LT.N) GO TO 100
   C COMPUTE CHI-SQUARE STATISTIC.
         NF=T-D+1
         DO 200 I=D1,T1
   200   COUNT(I-1)=COUNT(I)
         DO 300 I=D,T
   300   COUNT(I)=COUNT(I)**2
         TMIN1=T-1
         DO 400 R=D,TMIN1
         V=FLOAT(NN)/D**R
         V=V*ISTIR(R-1,D-1)
   400   COUNT(R)=COUNT(R)/V
         R=T
         V=FLOAT(NN)/D**(R-1)
```

OUTPUT FROM IRAND

STARTING NUMBER	1	DEGREES OF FREEDOM	4	
CHI-SQUARE STATISTIC	1.342	STATISTIC SHOULD BE GREATER THAN THIS VALUE		85.42 PERCENT OF THE TIME
STARTING NUMBER	11	DEGREES OF FREEDOM	4	
CHI-SQUARE STATISTIC	1.024	STATISTIC SHOULD BE GREATER THAN THIS VALUE		90.61 PERCENT OF THE TIME
STARTING NUMBER	1234	DEGREES OF FREEDOM	4	
CHI-SQUARE STATISTIC	1.362	STATISTIC SHOULD BE GREATER THAN THIS VALUE		85.08 PERCENT OF THE TIME
STARTING NUMBER	2	DEGREES OF FREEDOM	4	
CHI-SQUARE STATISTIC	1.322	STATISTIC SHOULD BE GREATER THAN THIS VALUE		85.76 PERCENT OF THE TIME
STARTING NUMBER	4444444	DEGREES OF FREEDOM	5	
CHI-SQUARE STATISTIC	1.291	STATISTIC SHOULD BE GREATER THAN THIS VALUE		93.58 PERCENT OF THE TIME
STARTING NUMBER	5643789	DEGREES OF FREEDOM	5	
CHI-SQUARE STATISTIC	5.900	STATISTIC SHOULD BE GREATER THAN THIS VALUE		31.60 PERCENT OF THE TIME
STARTING NUMBER	3400000	DEGREES OF FREEDOM	5	
CHI-SQUARE STATISTIC	5.339	STATISTIC SHOULD BE GREATER THAN THIS VALUE		37.59 PERCENT OF THE TIME
STARTING NUMBER	111006509	DEGREES OF FREEDOM	5	
CHI-SQUARE STATISTIC	3.728	STATISTIC SHOULD BE GREATER THAN THIS VALUE		58.93 PERCENT OF THE TIME
STARTING NUMBER	346777777	DEGREES OF FREEDOM	6	
CHI-SQUARE STATISTIC	2.128	STATISTIC SHOULD BE GREATER THAN THIS VALUE		90.76 PERCENT OF THE TIME

CONTINUED

STARTING NUMBER 89 DEGREES OF FREEDOM 6
CHI-SQUARE STATISTIC 15.755 STATISTIC SHOULD BE GREATER THAN THIS VALUE 1.51 PERCENT OF THE TIME

STARTING NUMBER 123456 DEGREES OF FREEDOM 6
CHI-SQUARE STATISTIC 7.650 STATISTIC SHOULD BE GREATER THAN THIS VALUE 26.49 PERCENT OF THE TIME

STARTING NUMBER 66665555 DEGREES OF FREEDOM 6
CHI-SQUARE STATISTIC 2.078 STATISTIC SHOULD BE GREATER THAN THIS VALUE 91.23 PERCENT OF THE TIME

OUTPUT FROM MRAND

STARTING NUMBER	1	DEGREES OF FREEDOM	4	
CHI-SQUARE STATISTIC	.494	STATISTIC SHOULD BE GREATER THAN THIS VALUE		97.41 PERCENT OF THE TIME
STARTING NUMBER	11	DEGREES OF FREEDOM	4	
CHI-SQUARE STATISTIC	1.180	STATISTIC SHOULD BE GREATER THAN THIS VALUE		88.14 PERCENT OF THE TIME
STARTING NUMBER	1234	DEGREES OF FREEDOM	4	
CHI-SQUARE STATISTIC	.518	STATISTIC SHOULD BE GREATER THAN THIS VALUE		97.17 PERCENT OF THE TIME
STARTING NUMBER	2	DEGREES OF FREEDOM	4	
CHI-SQUARE STATISTIC	1.144	STATISTIC SHOULD BE GREATER THAN THIS VALUE		88.72 PERCENT OF THE TIME
STARTING NUMBER	4444444	DEGREES OF FREEDOM	5	
CHI-SQUARE STATISTIC	5.433	STATISTIC SHOULD BE GREATER THAN THIS VALUE		36.54 PERCENT OF THE TIME
STARTING NUMBER	5643789	DEGREES OF FREEDOM	5	
CHI-SQUARE STATISTIC	1.701	STATISTIC SHOULD BE GREATER THAN THIS VALUE		88.87 PERCENT OF THE TIME
STARTING NUMBER	34000000	DEGREES OF FREEDOM	5	
CHI-SQUARE STATISTIC	3.060	STATISTIC SHOULD BE GREATER THAN THIS VALUE		69.08 PERCENT OF THE TIME
STARTING NUMBER	111006509	DEGREES OF FREEDOM	5	
CHI-SQUARE STATISTIC	1.837	STATISTIC SHOULD BE GREATER THAN THIS VALUE		87.12 PERCENT OF THE TIME
STARTING NUMBER	346777777	DEGREES OF FREEDOM	6	
CHI-SQUARE STATISTIC	2.775	STATISTIC SHOULD BE GREATER THAN THIS VALUE		83.65 PERCENT OF THE TIME
STARTING NUMBER	89	DEGREES OF FREEDOM	6	
CHI-SQUARE STATISTIC	2.323	STATISTIC SHOULD BE GREATER THAN THIS VALUE		88.77 PERCENT OF THE TIME

CONTINUED

STARTING NUMBER 123456 DEGREES OF FREEDOM 6
CHI-SQUARE STATISTIC 6.562 STATISTIC SHOULD BE GREATER THAN THIS VALUE 36.33 PERCENT OF THE TIME

STARTING NUMBER 66665555 DEGREES OF FREEDOM 6
CHI-SQUARE STATISTIC 4.474 STATISTIC SHOULD BE GREATER THAN THIS VALUE 61.29 PERCENT OF THE TIME

```
PA0.KF1
      C THIS FUNCTION COUNTS THE RELATIVE ORDERINGS OF A GROUP OF
      C ELEMENTS (U(1),.....,U(N)).
      C**********************************************************
      C U IS AN ARRAY OF REAL NUMBERS.(INPUT)
      C IT IS THE SIZE OF THE ARRAY.(INPUT).
      C AT THE END OF THE COMPUTATION FUNCTION KF RETURNS INTO THE
      C MAIN PROGRAM THE NUMBER OF THE RELATIVE ORDERINGS AMONG
      C THE ELEMENTS OF THE ARRAY U.
            FUNCTION KF(U,IT)
            INTEGER R,S,PREVL,C
            DIMENSION U(IT),C(10)
            R=IT
            PREVL=1
       1    CALL MAXIM(U,R,S)
            C(R)=S-1
            TEMP=U(S)
            U(S)=U(R)
            U(R)=TEMP
            R=R-1
            IF(R.GT.0) GO TO 1
            KF=C(IT)
       2    L=IT
            KF=KF+L*PREVL*C(IT-1)
            PREVL=PREVL*L
            IT=IT-1
            IF(IT.EQ.1) GO TO 3
            GO TO 2
       3    CONTINUE
            RETURN
            END
```

AQ.COMBSUB

```
      C THIS FUNCTION CALCULATES N FACTORIAL WHERE N IS AN INTEGER.
      C*************************************************************
      C N IS AN INTEGER.(INPUT)
      C L=IPROD(N) IS THE VALUE OF N FACTORIAL.
            FUNCTION IPROD(N)
            IF(N.EQ.1.OR.N.EQ.0) GO TO 4
            IPROD=1
            I=1
        1   I=I+1
            IF(I-N) 2,2,3
        2   IPROD=IPROD*I
            GO TO 1
        3   CONTINUE
            RETURN
        4   IPROD=1
            RETURN
            END
```

).MINI

```
            SUBROUTINE MININ(U,N,IS)
            DIMENSION U(N)
            I=1
            SMIN=U(1)
            IS=1
        1   I=I+1
            IF(I.GT.N) GO TO 2
            IF(SMIN.LT.U(I)) GO TO 1
            SMIN=U(I)
            IS=I
            GO TO 1
        2   CONTINUE
            RETURN
            END
```

).MAXI

```
      C SUBROUTINE MAXIM FINDS THE POSITION OF THE MAXIMUM OF T
      C ELEMENTS, U(1),U(2),....,U(T).
      C*************************************************************
      C U IS AN ARRAY OF REAL NUMBERS,WHICH WILL BE SEARCHED TO FIND THE
      C POSITION OF THE MAXIMUM ELEMENT IN IT.(INPUT)
      C IT IS THE SIZE OF THE ARRAY U.(INPUT)
      C IS IS THE POSITION OF THE MAXIMUM ELEMENT OF THE ARRAY U.(OUTPUT).
            SUBROUTINE MAXIM(U,IT,IS)
            DIMENSION U(10)
            I=1
            SMAX=U(1)
            IS=1
        1   I=I+1
            IF(I.GT.IT) GO TO 2
            IF(SMAX.GT.U(I)) GO TO 1
            SMAX=U(I)
            IS=I
            GO TO 1
        2   CONTINUE
            RETURN
            END
```

AQ.COMBSUB

```
      C THIS FUNCTION CALCULATES N FACTORIAL,WHERE N IS AN INTEGER.
      C*************************************************************
      C N IS AN INTEGER.(INPUT)
      C L=IPROD(N) IS THE VALUE OF N FACTORIAL.
            FUNCTION IPROD(N)
            IF(N.EQ.1.OR.N.EQ.0) GO TO 4
            IPROD=1
            I=1
        1   I=I+1
            IF(I-N) 2,2,3
        2   IPROD=IPROD*I
            GO TO 1
        3   CONTINUE
            RETURN
        4   IPROD=1
            RETURN
            END
```

).MINI

```
            SUBROUTINE MINIM(U,N,IS)
            DIMENSION U(N)
            I=1
            SMIN=U(1)
            IS=1
        1   I=I+1
            IF(I.GT.N) GO TO 2
            IF(SMIN.LT.U(I)) GO TO 1
            SMIN=U(I)
            IS=I
            GO TO 1
        2   CONTINUE
            RETURN
            END
```

).MAXI

```
      C SUBROUTINE MAXIM FINDS THE POSITION OF THE MAXIMUM OF T
      C ELEMENTS, U(1),U(2),....,U(T).
      C*************************************************************
      C U IS AN ARRAY OF REAL NUMBERS,WHICH WILL BE SEARCHED TO FIND THE
      C POSITION OF THE MAXIMUM ELEMENT IN IT.(INPUT)
      C IT IS THE SIZE OF THE ARRAY U.(INPUT)
      C IS IS THE POSITION OF THE MAXIMUM ELEMENT OF THE ARRAY U.(OUTPUT).
            SUBROUTINE MAXIM(U,IT,IS)
            DIMENSION U(10)
            I=1
            SMAX=U(1)
            IS=1
        1   I=I+1
            IF(I.GT.IT) GO TO 2
            IF(SMAX.GT.U(I)) GO TO 1
            SMAX=U(I)
            IS=I
            GO TO 1
        2   CONTINUE
            RETURN
            END
```

```
FO*PA0,TEST6
  1       C PERMUTATION TEST.
  2             INTEGER T,ORD(6000)
  3             REAL U(10)
  4             READ(8,1500) N
  5       1500  FORMAT(I3)
  6             WRITE(5,11)
  7        11   FORMAT(1H1,'OUTPUT FROM IRAND'///)
  8             DO 150 NTEST=1,N
  9             READ(8,10) N,T,MS
 10        10   FORMAT(I8,I3,I12)
 11             IVAL=MS
 12             NG=N/T
 13             SUM=0.
 14       C FUNCTION IPROD CALCULATES T FACTORIAL. NN DENOTES THE NUMBER
 15       C OF CATEGORIES.
 16             NN=IPROD(T)
 17             DO 100 I=1,NN
 18       C THE ARRAY ORD DENOTES THE NN CATEGORIES.
 19        100  ORD(I)=0
 20             CALL PINAX(MS)
 21             KOUNT=0
 22       C THE INPUT SEQUENCE IS DEVIDED INTO NG GROUPS,AND THE NUBER OF TIMES
 23       C EACH ORDERING APPEARS IS COUNTED.
 24        3    I=1
 25        1    U(I)=IRAND(MS)/(2.**35)
 26             I=I+1
 27             IF(I.GT.T) GO TO 2
 28             GO TO 1
 29        2    KOUNT=KOUNT+1
 30             IF(KOUNT.EQ.NG) GO TO 4
 31             IT=T
 32             L=KF(U,IT)
 33             L=L+1
 34             ORD(L)=ORD(L)+1
 35             GO TO 3
 36        4    IT=T
 37             L=KF(U,IT)
 38             L=L+1
 39             ORD(L)=ORD(L)+1
 40       C COMPUTATION OF THE CHI-SQUARE STATISTIC.
 41             DO 5 I=1,NN
 42        5    ORD(I)=ORD(I)**2
 43             PP=1./FLOAT(NN)
 44             NF=NN-1
 45             DO 6 I=1,NN
 46        6    SUM=SUM+ORD(I)
 47             VV=SUM/PP
 48             VV=VV/FLOAT(NG)
 49             VV=VV-FLOAT(NG)
 50             VAR=CHI(VV,NF,$75)
 51             VAR=VAR*100.
 52             VAR=100.-VAR
 53             WRITE(5,1000) IVAL,NN,VV,VAR
 54        1000 FORMAT(10X,'STARTING NUMBER',1X,I12,1X,'NUMBER OF CATEGORIES',2X,
 55            1I3/10X,'CHI-SQUARE STATISTIC',1X,F7.3,1X,'STATISTIC SHOULD BE GREA
 56            2TER THAN THIS VALUE',1X,F8.2,1X,'PER CENT OF THE TIME'//)
```

CONTINUED

```
57                GO TO 150
58          75    WRITE(5,80)
59          80    FORMAT('1H1,'OVER. OCCURED IN FUNCTION CHI')
60          150   CONTINUE
61                END
```

OUTPUT FROM IRAND

STARTING NUMBER	11	NUMBER OF CATEGORIES	6		
CHI-SQUARE STATISTIC	5.240	STATISTIC SHOULD BE GREATER THAN THIS VALUE		38.73	PER CENT OF THE TIME
STARTING NUMBER	345	NUMBER OF CATEGORIES	6		
CHI-SQUARE STATISTIC	5.360	STATISTIC SHOULD BE GREATER THAN THIS VALUE		37.35	PER CENT OF THE TIME
STARTING NUMBER	34567890	NUMBER OF CATEGORIES	6		
CHI-SQUARE STATISTIC	3.680	STATISTIC SHOULD BE GREATER THAN THIS VALUE		59.63	PER CENT OF THE TIME
STARTING NUMBER	123409	NUMBER OF CATEGORIES	6		
CHI-SQUARE STATISTIC	4.280	STATISTIC SHOULD BE GREATER THAN THIS VALUE		50.98	PER CENT OF THE TIME
STARTING NUMBER	1	NUMBER OF CATEGORIES	24		
CHI-SQUARE STATISTIC	22.760	STATISTIC SHOULD BE GREATER THAN THIS VALUE		47.49	PER CENT OF THE TIME
STARTING NUMBER	2	NUMBER OF CATEGORIES	24		
CHI-SQUARE STATISTIC	16.280	STATISTIC SHOULD BE GREATER THAN THIS VALUE		84.30	PER CENT OF THE TIME
STARTING NUMBER	33	NUMBER OF CATEGORIES	24		
CHI-SQUARE STATISTIC	18.800	STATISTIC SHOULD BE GREATER THAN THIS VALUE		71.28	PER CENT OF THE TIME
STARTING NUMBER	1111	NUMBER OF CATEGORIES	24		
CHI-SQUARE STATISTIC	14.360	STATISTIC SHOULD BE GREATER THAN THIS VALUE		91.61	PER CENT OF THE TIME
STARTING NUMBER	1443256	NUMBER OF CATEGORIES	120		
CHI-SQUARE STATISTIC	101.440	STATISTIC SHOULD BE GREATER THAN THIS VALUE		87.61	PER CENT OF THE TIME
STARTING NUMBER	12300098	NUMBER OF CATEGORIES	120		
CHI-SQUARE STATISTIC	135.424	STATISTIC SHOULD BE GREATER THAN THIS VALUE		14.41	PER CENT OF THE TIME

OUTPUT FROM MRAND

STARTING NUMBER	11	NUMBER OF CATEGORIES	6		
CHI-SQUARE STATISTIC	4.400	STATISTIC SHOULD BE GREATER THAN THIS VALUE		49.34	PER CENT OF THE TIME
STARTING NUMBER	345	NUMBER OF CATEGORIES	6		
CHI-SQUARE STATISTIC	8.840	STATISTIC SHOULD BE GREATER THAN THIS VALUE		11.56	PER CENT OF THE TIME
STARTING NUMBER	34567890	NUMBER OF CATEGORIES	6		
CHI-SQUARE STATISTIC	13.520	STATISTIC SHOULD BE GREATER THAN THIS VALUE		1.90	PER CENT OF THE TIME
STARTING NUMBER	123409	NUMBER OF CATEGORIES	6		
CHI-SQUARE STATISTIC	10.400	STATISTIC SHOULD BE GREATER THAN THIS VALUE		6.47	PER CENT OF THE TIME
STARTING NUMBER	1	NUMBER OF CATEGORIES	24		
CHI-SQUARE STATISTIC	29.960	STATISTIC SHOULD BE GREATER THAN THIS VALUE		15.06	PER CENT OF THE TIME
STARTING NUMBER	2	NUMBER OF CATEGORIES	24		
CHI-SQUARE STATISTIC	21.920	STATISTIC SHOULD BE GREATER THAN THIS VALUE		52.51	PER CENT OF THE TIME
STARTING NUMBER	33	NUMBER OF CATEGORIES	24		
CHI-SQUARE STATISTIC	16.160	STATISTIC SHOULD BE GREATER THAN THIS VALUE		84.83	PER CENT OF THE TIME
STARTING NUMBER	1111	NUMBER OF CATEGORIES	24		
CHI-SQUARE STATISTIC	25.880	STATISTIC SHOULD BE GREATER THAN THIS VALUE		30.66	PER CENT OF THE TIME
STARTING NUMBER	1443256	NUMBER OF CATEGORIES	120		
CHI-SQUARE STATISTIC	101.440	STATISTIC SHOULD BE GREATER THAN THIS VALUE		87.61	PER CENT OF THE TIME
STARTING NUMBER	12300098	NUMBER OF CATEGORIES	120		
CHI-SQUARE STATISTIC	116.512	STATISTIC SHOULD BE GREATER THAN THIS VALUE		54.74	PER CENT OF THE TIME

CONTINUED

STARTING NUMBER 44445566 NUMBER OF CATEGORIES 120
CHI-SQUARE STATISTIC 122.176 STATISTIC SHOULD BE GREATER THAN THIS VALUE 40.24 PER CENT OF THE TIME

STARTING NUMBER 4456 NUMBER OF CATEGORIES 120
CHI-SQUARE STATISTIC 100.624 STATISTIC SHOULD BE GREATER THAN THIS VALUE 88.78 PER CENT OF THE TIME

C34

```
PAO.TEST3
      C          KOLMOGOROV-SMIRNOF TEST.
      C THIS TEST EXAMINES THE RANDOMNESS OF THE RANDSUB2 SUBROUTINE
      C CONSIDERING THE RANDOM NUMBERS AS REAL NUMBERS BETWEEN 0 AND 1.
            INTEGER NSTART(20)
            REAL KPLUSN(20),KMINN(20),XX(9),YY(9)
            DIMENSION X(1000),Q(1000),Q1(1000),R(20),R1(20),RKS(2)
            DIMENSION YE(20),YE1(20)
      C DATA FOR SUBROUTINE INTER.
            DATA (XX(I),I=1,6)/.02,.0661,.1557,.3746,.584,.8275/
            DATA (XX(I),I=7,9)/1.2185,1.5111,2./
            DATA (YY(I),I=1,9)/99.92,99.,95.,75.,50.,25.,5.,1.,.0003/
            SR=SQRT(1000.)
            SR1=SQRT(20.)
            WRITE(5,13)
      13    FORMAT(1H1,'O U T P U T F R O M I R A N D'//)
            DO 150 NTEST=1,20
            READ(8,10) MS
      10    FORMAT(I12)
      C SEED FOR THE GENERATOR IRAND.
            NSTART(NTEST)=MS
            CALL PINAX(MS)
      C THIS DO LOOP GENERATES 1000 UNIFORMLY DISTRIBUTED NUMBERS BETWEEN 0,1
            DO 11I=1,1000
      11    X(I)=IRAND(MS)/(2.**35)
      C THE NUMBERS ARE SORTED IN ASCENDING ORDER.
            CALL SORTRL(1000,X)
      C CALCULATION OF THE +K1000,-K1000 STATISTIC.
            DO 12 J=1,1000
            Q(J)=(FLOAT(J)/1000.)-X(J)
      12    Q1(J)=X(J)-(FLOAT(J-1)/1000.)
            J=1
            RMAX1=Q(J)
            RMAX2=Q1(J)
      17    RMAX1=AMAX1(RMAX1,Q(J+1))
            RMAX2=AMAX1(RMAX2,Q1(J+1))
            J=J+1
            IF(J.EQ.1000) GO TO 18
            GO TO 17
      18    CONTINUE
            KPLUSN(NTEST)=SR*RMAX1
            KMINN(NTEST)=SR*RMAX2
      150   CONTINUE
      C WE INTERPOLATE THE KOL-SMIRNOV TABLE TO FIND THE PERCENTAGE OF TIME T
      C THE STATISTIC +K1000 OR -K1000 IS GREATER THAN THE KS STATISTIC.
            CALL INTER(XX,YY,KPLUSN,YE,9,20,$5,$6)
            CALL INTER(XX,YY,KMINN,YE1,9,20,$5,$6)
            GO TO 20
      5     WRITE(5,981)
      981   FORMAT(1X,'ERROR. INPUT VALUE X LESS THAN SMALLEST X ON CURVE.')
            GO TO 20
      6     WRITE(5,982)
      982   FORMAT(1X,'ERROR. INPUT VALUE IS LARGER THAN LARGEST X ON CURVE')
      20    CONTINUE
            WRITE(5,19)
      19    FORMAT(1X,'STARTING NUMBER',5X,'KPLUSN STATISTIC',10X,'PROBABILIT
           1Y THAT STATISTIC BE GREATER THAN KPLUSN VALUE'//)
```

```
      WRITE(5,22) (NSTART(I),KPLUSN(I),YE(I),I=1,20)
   22 FORMAT(1X,I12,7X,F10.6,40X,F10.3/)
      WRITE(5,42)
   42 FORMAT(1X,'STARTING NUMBER',5X,'KMINN STATISTIC',10X,'PROBABILITY
     1 THAT STATISTIC BE GREATER THAN KMINN VALUE'//)
      WRITE(5,63) (NSTART(I),KMINN(I),YE(I),I=1,20)
   63 FORMAT(1X,I12,10X,F10.6,20X,F10.3)
C PROCEDURE TO FIND THE KS STATISTIC OF THE VALUES +K1000,-K1000.
      CALL SORTRL(20,KPLUSN)
      CALL SORTRL(20,KMINN)
      DO 21 I=1,20
      Q(I)=(FLOAT(I)/20.-1.+EXP(-2.*KPLUSN(I)**2))
      Q1(I)=(1.-EXP(-2.*KPLUSN(I)**2)-FLOAT(I-1)/20.)
      R(I)=(FLOAT(I)/20.-1.+EXP(-2.*KMINN(I)**2))
   21 R1(I)=(1.-EXP(-2.*KMINN(I)**2)-FLOAT(I-1)/20.)
      J=1
      RMAX1=Q(J)
      RMAX2=Q1(J)
   43 RMAX1=AMAX1(RMAX1,Q(J+1))
      RMAX2=AMAX1(RMAX2,Q1(J+1))
      J=J+1
      IF(J.EQ.20) GO TO 44
      GO TO 43
   44 CONTINUE
      RKS(1)=SR1*RMAX1
      RKS(2)=SR1*RMAX2
      CALL INTER(XX,YY,RKS,YE,9,2,$5,$6)
      WRITE(5,64)
   64 FORMAT(1X,'VALUE OF K+20',3X,'PROB. THAT STAT. BE GREATER THAN K+2
     1 0 VALUE',3X,'VALUE OF K-20',3X,'PROB THAT STAT. BE GREATER THAN K-
     220 VALUE'//)
      WRITE(5,45) (RKS(I),YE(I),I=1,2)
   45 FORMAT(1X,F10.6,14X,F10.3,30X,F10.6,17X,F10.3)
      K=1
      RMAX1=R(K)
      RMAX2=R1(K)
   57 RMAX1=AMAX1(RMAX1,R(K+1))
      RMAX2=AMAX1(RMAX2,R1(K+1))
      K=K+1
      IF(K.EQ.20) GO TO 58
      GO TO 57
   58 CONTINUE
      RKS(1)=SR1*RMAX1
      RKS(2)=SR1*RMAX2
      CALL INTER(XX,YY,RKS,YE,9,2,$5,$6)
      WRITE(5,88)
   88 FORMAT(1X,'VALUE OF K+(-20)',2X,'PROB THAT STAT BE GREATER THAN K+
     1(-20)',3X,'VALUE OF K-(-20)',3X,'PROB THAT STAT. BE GREATER THAN K-
     2(-20)'//)
      WRITE(5,89) (RKS(I),YE(I),I=1,2)
   89 FORMAT(1X,F10.6,14X,F10.3,30X,F10.6,17X,F10.3)
      END
```

OUTPUTFROMIRAND

STARTING NUMBER	KPLUSN STATISTIC	PROBABILITY THAT STATISTIC BE GREATER THAN KPLUSN VALUE
2398	.579953	50,483
129054783	.781584	29,714
765	.641300	44,117
99984567	.762550	31,668
2000000001	.664023	41,784
9153	.316974	80,265
2	.730920	34,916
7	.636055	44,656
2900	.564947	52,275
2016	.113795	96,871
54091	.686070	39,521
2934	.888293	21,890
19850043	.289774	82,750
723	.292042	82,543
2231974	.614424	46,876
56	.221994	88,943
2983330917	1.246712	4,587
40000	.327369	79,315
666666666	.605538	47,789

CONTINUED

298656	.707178	37,353

STARTING NUMBER	KMINN STATISTIC	PROBABILITY THAT STATISTIC BE GREATER THAN KMINN VALUE
2398	.772372	30.660
129054783	.723018	35.727
765	.755243	32.419
99984567	.463091	64.435
2000000001	.452278	65.726
9153	.510357	58.792
2	.190439	91.826
7	.782211	29.650
2900	.385453	73.704
20816	.920320	20.252
54091	.349950	77.252
2934	.580093	50.467
19850043	.644510	43.787
723	1.031154	14.583
2231974	.533032	56.085
56	.548481	54.241
4983330917	.388906	73.292
40000	.698929	38.200
666666666	.504252	81.427
298656	.432051	68.141

VALUE OF K+20	PROB. THAT STAT. BE GREATER THAN K+20 VALUE	VALUE OF K-20	PROB THAT STAT. BE GREATER THAN K-20
.870793	22.786	.768433	31.064

VALUE OF K+(-20)	PROB THAT STAT BE GREATER THAN K+(-20)	VALUE OF K-(-20)	PROB THAT STAT. BE GREATER THAN K-(-20)
.868209	22.918	.532231	56.181

OUTPUT FROM RAND

STARTING NUMBER	KPLUSN STATISTIC	PROBABILITY THAT STATISTIC BE GREATER THAN KPLUSN VALUE
2398	.437100	67.538
129054783	.864976	23.083
765	.771790	30.720
99984567	1.024276	14.935
2000000001	.797697	28.060
9153	.151149	95.203
2	.694112	38.695
7	1.027913	14.749
2900	.136559	95.854
20816	.454146	65.503
54091	.516904	58.011
2934	1.292888	3.983
19850043	.219485	89.172
723	.852282	23.732
2231974	.563627	52.432
56	.150823	95.218
2983330917	1.496133	1.205
40000	.672901	40.873
666666666	.714385	36.613
298656	1.187520	6.585

CONTINUED

STARTING NUMBER	KMINN STATISTIC	PROBABILITY THAT STATISTIC BE GREATER THAN KMINN VALUE
2398	.386293	73.604
129054783	.138131	95.784
765	.368655	75.543
99984567	.102569	97.372
2000000001	.696363	38.464
9153	1.731965	.548
2	.475504	62.953
7	.244698	86.869

2900	.603211	48.028	
20616	.758438	32.091	
54091	.621263	46.174	
2934	.188801	91.976	
19850043	.524406	57.115	
723	.256701	85.772	
2231974	1.488393	1.310	
56	.687669	39.356	
2983330917	.415509	70.116	
40000	.301203	81.706	
666666666	.403365	71.566	
298656	.157696	94.818	

VALUE OF K+20	PROB. THAT STAT. BE GREATER THAN K+20 VALUE	VALUE OF K-20	PROB THAT STAT. BE GREATER THAN K-20
.433654	61.980	.875184	22.561

VALUE OF K+(-20)	PROB THAT STAT BE GREATER THAN K+(-20)	VALUE OF K-(-20)	PROB THAT STAT. BE GREATER THAN K-(-20)
1.153858	8.306	.393964	72.688

```
PAO.INTER1
      C       LINEAR INTERPOLATION.
      C*****************************************************************
      C X AND Y ARE ARRAYS OF THE PAIRS (X(N),Y(N)) WHICH REPRESENT POINTS
      C ON THE CURVE F(X). (INPUT)
      C XE IS AN ARRAY OF POINTS WHERE XE IS LARGER THAN X(N-1) AND
      C SMALLER THAN X(N) FOR N=2,3,...,N (INPUT)
      C YE IS AN ARRAY OF THE CORRESPONDING INTERPOLATED VALUES OF THE
      C POINTS XE.(OUTPUT)
      C N IS THE SIZE OF THE ARRAY X OR Y. (INPUR)
      C N1 IS THE SIZE OF THE ARRAY XE OR YE.(INPUT)
            SUBROUTINE INTER(X,Y,XE,YE,N,N1,*,*)
            DIMENSION X(N),Y(N),XE(N1),YE(N1)
            I=1
      C THIS STATEMENT CHEKS FOR X VALUES LESS THAN SMALLEST X
      C  ON CURVE.
      100   IF(XE(I).LT.X(1)) RETURN 7
            J=2
      C THE FOLLOWING IF STATEMENT COMPARES THE GIVEN X VALUE
      C WITH SUCCESSIVE VALUES ON THE CURVE LOOKING FOR AN X
      C VALUE THAT IS GREATER THAN THE GIVEN VALUE.
      2     IF(XE(I)-X(J)) 112,111,110
      C NOT FOUND YET.
      110   J=J+1
      C CHEK WHETHER GIVEN X VALUE IS LARGER THAN LARGEST
      C X ON CURVE.
            IF(J.GT.N) RETURN 8
            GO TO 2
      C EQUAL
      111   YE(I)=Y(J)
            GO TO 200
      C INTERPOLATE
      112   YE(I)=Y(J-1)+(Y(J)-Y(J-1))/(X(J)-X(J-1))*(XE(I)-X(J-1))
      200   I=I+1
            IF(I.GT.N1) GO TO 150
            GO TO 100
      150   CONTINUE
            RETURN
            END
```

```
PAO.NORMAL1
      C GENERATION OF NORMALY DISTRIBUTED RANDOM-NUMBERS HAVING SPECIFIED
      C MEAN AND STANDARD DEVIATION.
      C*****************************************************************
      C X IS THE ARRAY OF RANDOM NUMBERS (OUTPUT).
      C N IS THE NUMBER OF RANDOM NUMBERS DESIRED (INPUT).
      C EX IS THE MEAN OF RANDOM NUMBERS(INPUT)
      C STDX IS THE ST. DEVIATION OF THE RANDOM NUMBERS(INPUT).
      C MS IS THE STARTING VALUE(INTEGER) OF THE SUBROUTINE PINAX.
      C MS SHOULD BE CHOSEN IN THE INTERVAL (0,2**35-1).DIFFERENT
      C VALUES OF MS WILL RESULT IN DIFFERENT OUTPUT RANDOM SEQUENCES.
      C AT THE OF THE COMPUTATION MS CONTAINS THE LAST NUMBER GENERA-
      C TED BY IRAND (INPUT,OUTPUT).
            SUBROUTINE RANDNR(X,N,EX,STDX,MS)
            DIMENSION R(12)
            DIMENSION X(N)
            CALL PINAX(MS)
            ENTRY FASTNR(X,N,EX,STDX,MS)
            DO 1 I=1,N
            SUM=0.
            DO 2 J=1,12
            R(J)=IRAND(MS)/(2.**35)
      2     SUM=SUM+R(J)
            X(I)=STDX*(SUM-6.)+EX
      1     CONTINUE
            RETURN
            END
```

```
F0*PA0.TEST11
  1       C MAXIMUM OF T TEST.
  2             DIMENSION U(10),V(1000),Q(1000),R(1000),YE(100),YE1(100),XX(9),
  3            1YY(9)
  4             REAL KPLUSN(100),KMINN(100)
  5             INTEGER T
  6             DATA (XX(I),I=1,6)/.0001,.0661,.1557,.3746,.584,.8275/
  7             DATA (XX(I),I=7,9)/1.2185,1.5111,2./
  8             DATA (YY(I),I=1,9)/100.,99.,95.,75.,50.,25.,5.,1.,.0003/
  9             READ (8,1) M,N,T
 10        1    FORMAT(I4,I8,I3)
 11             WRITE(5,1000) T
 12        1000 FORMAT(1H1,'MAXIMUM OF',3X,I3,2X,'TEST'//)
 13             WRITE(5,11)
 14        11   FORMAT(1X,'OUTPUT FROM IRAND'///)
 15             SQ=SQRT(FLOAT(N))
 16             DO 200 NTEST=1,M
 17             READ(8,2) MS
 18        2    FORMAT(I12)
 19             CALL PINAX(MS)
 20       C THE INPUT SEQUENCE IS DIVIDED INTO N GROUPS OF T ELEMENTS EACH
 21             DO 3 I=1,N
 22             DO 4 J=1,T
 23             U(J)=IRAND(MS)/2.**35
 24        4    CONTINUE
 25       C WE FIND THE MAXIMUM ELEMENT OF EACH GROUP.
 26             L=1
 27             RMAX1=U(L)
 28        5    RMAX1=AMAX1(RMAX1,U(L+1))
 29             L=L+1
 30             IF(L.EQ.T) GO TO 6
 31             GO TO 5
 32        6    CONTINUE
 33       C * V * DENOTES THE MAXIMUM ELEMENT OF A GROUP.
 34             V(I)=RMAX1
 35        3    CONTINUE
 36       C THE NUMBERS V ARE SORTED INTO ASCENDING ORDER.
 37             CALL SORTRL(N,V)
 38       C COMPUTATION OF THE KS STATISTIC.
 39             DO 7 I=1,N
 40             Q(I)=FLOAT(I)/N-V(I)**T
 41             R(I)=V(I)**T-FLOAT(I-1)/FLOAT(N)
 42        7    CONTINUE
 43             I=1
 44             RMAX1=Q(I)
 45             RMAX2=R(I)
 46        8    RMAX1=AMAX1(RMAX1,Q(I+1))
 47             RMAX2=AMAX1(RMAX2,R(I+1))
 48             I=I+1
 49             IF(I.EQ.N) GO TO 9
 50             GO TO 8
 51        9    CONTINUE
 52             KPLUSN(NTEST)=SQ*RMAX1
 53             KMINN(NTEST)=SQ*RMAX2
 54        200  CONTINUE
 55       C INTERPOLATION OF THE TABLE
 56             CALL INTER(XX,YY,KPLUSN,YE,9,NTEST,$25,$75)
```

CONTINUED

```
      CALL INTER(XX,YY,KMINN,YE1,9,NTEST,$25,$75)
      GO TO 100
   25 WRITE(5,981)
  981 FORMAT(1X,'ERROR. INPUT VALUE X LESS THAN SMALLEST X ON CURVE')
      GO TO 201
   75 WRITE(5,982)
  982 FORMAT(1X,'ERROR. INPUT VALUE X LARGER THAN LARGEST X ON CURVE')
      GO TO 201
  100 CONTINUE
      WRITE(5,150)
  150 FORMAT(4X,'K P L U S N',20X,'PROBABILITY THAT KPLUSN IS GREATER THA
     1N  KS',//)
      WRITE(5,151)  (KPLUSN(I),YE(I),I=1,NTEST)
  151 FORMAT(1X,F9.4,25X,F9.1)
      WRITE(5,250)
  250 FORMAT(4X,'K M I N N',20X,'PROBABILITY THAT KMINN IS GREATER THAN
     1KS',//)
      WRITE(5,251) (KMINN(I),YE1(I),I=1,NTEST)
  251 FORMAT(1X,F9.4,25X,F9.1)
  201 END
```

MAXIMUM OF 2 TEST

OUTPUT FROM IRAND

K P L U S N	PROBABILITY THAT KPLUSN IS GREATER THAN KS
.6449	43.7
.5668	52.1
.9702	17.7
.9565	65.2
1.0364	14.3
.7527	32.7
.4862	61.7
.1646	94.2
.9993	16.2
.4976	60.3
.3109	80.8
.3856	73.7
.1529	95.1
.2573	85.7
.3884	73.3
.6968	38.4
.5095	58.9
.5446	54.7
.4606	64.7
.3169	80.3

K M I N N	PROBABILITY THAT KMINN IS GREATER THAN KS
.5871	49.7
.7782	30.1
.6287	45.4
.8337	24.7
.1107	97.0
.8234	25.4
.6522	43.0
.7454	33.4
.3732	75.1
.6257	45.7
.6327	45.0
.8882	21.9
.7597	32.0
.5611	52.7
1.2097	5.5
.2997	81.8
.9178	20.4
.7338	34.6
.6300	45.3
1.3699	2.9

C4D

MAXIMUM OF 2 TEST

OUTPUT FROM MRAND

K P L U S N	PROBABILITY THAT KPLUSN IS GREATER THAN K
.2229	88.9
.9997	16.2
.2099	90.1
.7560	32.3
1.0379	14.2
.9048	21.0
.5432	54.9
.3590	76.4
.9729	17.6
.8466	24.0
1.1609	7.9
.3438	77.8
.5275	56.7
.4322	68.1
.6553	42.7
.8171	26.1
.5945	48.9
.6805	40.1
1.3059	3.8
.7669	31.2

K M I N N	PROBABILITY THAT KMINN IS GREATER THAN KS
1.0293	14.7
.2514	86.3
1.0604	13.1
.3443	77.8
.1071	97.2
.3441	77.8
.4520	65.8
.8145	26.3
.4965	60.4
.7072	37.3
.3269	79.4
.7580	32.1
.6087	26.9
.7363	34.4
.3079	81.1
.1346	95.9
.4743	63.1
.5549	53.5
.2788	83.7
.9381	19.3

MAXIMUM OF 3 TEST

OUTPUT FROM IRAND

K P L U S N	PROBABILITY THAT KPLUSN IS GREATER THAN K
.3335	78.8
.3099	47.3
.4412	57.0
.6086	47.5
.4074	71.1
1.0995	11.1
.3763	74.8
.3396	78.2
.5116	58.6
.6044	47.9
.6781	48.3
.3052	81.3
.1700	93.6
1.1256	9.8
.5576	23.4
.6758	40.6
.6092	47.4
1.3712	2.9
.5061	59.3
.6158	46.7

K M I N N	PROBABILITY THAT KMINN IS GREATER THAN KS
.6886	32.3
.2179	89.3
.6176	46.5
.6812	40.0
.6217	46.1
.5628	52.5
.9284	19.8
.6386	44.4
.5798	50.5
.6599	42.2
.5453	54.6
.6779	40.4
1.3883	2.7
.3038	81.5
.2200	89.1
1.1729	7.3
.5920	42.2
.2323	88.0
.3669	75.7
.7077	37.3

MAXIMUM OF 3 TEST

OUTPUT FROM MRAND

K P L U S N	PROBABILITY THAT KPLUSN IS GREATER THAN K
1.3683	3.0
.6523	43.0
.2420	87.1
.8588	23.4
.7075	37.3
1.0378	14.2
.1995	91.0
.4713	63.4
.6120	47.1
.3533	76.9
.9772	17.3
.7562	32.3
.0375	99.4
.7284	35.2
.4769	62.8
.9817	17.1
.1396	95.7
.8120	26.6
.4493	66.1
.3758	74.9

K M I N N	PROBABILITY THAT KMINN IS GREATER THAN KS
.1129	96.9
.6041	47.9
.7611	31.8
.3281	79.2
.3332	78.8
.3869	73.5
1.1396	9.0
.9196	20.3
.5620	52.6
.4694	63.7
.5414	55.1
.3430	77.9
1.2004	5.9
.1966	91.3
.6275	45.5
.2428	87.0
.7564	32.3
.4737	63.2
.6758	40.6
.5640	52.4

MAXIMUM OF 4 TEST

OUTPUT FROM IRAND

K P L U S N	PROBABILITY THAT KPLUSN IS GREATER THAN KS
.4801	62.4
.7155	36.5
.4357	67.7
.2170	89.4
.1919	91.7
1.2506	4.6
.6286	45.4
.5163	58.1
.9202	20.3
.3538	76.9
.4770	62.8
.4444	66.7
.2283	88.4
1.2886	4.0
1.0556	13.3
.5711	51.5
.7289	35.1
1.2882	4.0
.4184	69.8
.3415	78.0

K M I N N	PROBABILITY THAT KMINN IS GREATER THAN KS
.6712	41.0
.4423	66.9
.8216	25.6
1.1739	7.3
.7056	37.5
.2051	90.5
.5630	52.5
.6781	40.3
.3684	75.6
1.1629	7.8
.6789	40.3
.5560	53.3
1.5136	1.0
.2756	84.0
.1915	91.7
1.0293	14.7
.3509	77.2
.2020	90.8
.6149	46.8
.7110	37.0

C44

MAXIMUM OF 4 TEST

OUTPUT FROM MRAND

K P L U S N	PROBABILITY THAT KPLUSN IS GREATER THAN KS
1.6788	.7
.4405	67.1
.4973	60.4
.7845	29.4
.9419	19.1
.9627	18.1
.2949	82.3
.2978	82.0
.4345	67.8
.6644	41.7
.8923	21.7
.4240	69.1
.0301	99.5
.3384	78.3
.1506	95.2
1.0967	11.2
.3593	76.4
.5416	55.1
.6607	42.1
.6323	45.0

K M I N N	PROBABILITY THAT KMINN IS GREATER THAN KS
.0581	99.1
.4956	60.6
1.0833	11.9
.2156	89.5
.3745	75.0
.4030	71.6
.8304	24.8
1.1343	9.3
.8552	23.6
.3009	81.7
.7813	29.7
.5292	56.5
1.5533	.9
.4607	64.7
.9572	18.4
.1011	97.4
.6912	39.0
.6771	40.4
.6404	44.2
.2803	83.6

MAXITION OF S TEST

OUTPUT FROM IRAND

K P L U S N	PROBABILITY THAT KPLUSN IS GREATER THAN KS
.4056	71.3
.7999	27.8
.4269	68.8
.3104	80.9
.4101	70.8
1.1283	9.6
1.1301	9.5
.4159	70.1
.3425	77.9
.6569	42.5
.3423	78.0
.2860	83.1
.3205	79.9
.9773	17.3
.7642	31.5
.4154	70.1
.7385	34.1
1.7030	.6
.4250	69.0
.4009	71.9

K M I N N	PROBABILITY THAT KMINN IS GREATER THAN KS
.5948	48.9
.3099	80.9
.9517	18.6
.7630	31.6
.8671	23.0
.2157	89.5
.3030	27.5
.7274	35.3
.6548	42.7
.6807	40.1
.7990	27.9
.7461	33.4
.6938	38.7
.0846	98.2
.2527	86.1
.6643	41.8
.4156	70.1
.1678	93.9
.6897	39.1
.7822	29.6

MAXIMUM OF 5 TEST

OUTPUT FROM MRAND

K P L U S N	PROBABILITY THAT KPLUSN IS GREATER THAN K
1.7063	.6
.3404	78.1
.2301	88.2
.8880	21.9
.6364	44.6
.7528	32.7
.4227	69.3
.3374	78.4
.4382	67.4
.5128	58.5
.9515	18.7
.3463	77.6
.0065	99.9
.3167	80.3
.3191	80.1
.8483	23.9
.3315	78.9
.6526	43.0
.4009	71.9
.5339	56.0

K M I N N	PROBABILITY THAT KMINN IS GREATER THAN KS
.3598	76.4
.8117	26.6
1.1078	10.7
.1698	93.7
.4350	67.8
.6018	48.2
.6451	43.7
1.2164	5.1
1.0290	14.7
.6500	43.2
.6936	38.7
.8132	26.5
1.3463	3.3
.3226	79.8
1.1430	8.9
.2162	89.5
.5004	50.4
.7216	35.9
.6377	44.5
.3490	77.3

```
*PA0.TEST9
1       C CERIAL CORRELATION TEST.
2             REAL U(20000),MI,NUMER,MD
3             READ(8,10) N
4       10    FORMAT(I4)                                          .033
5             WRITE(5,12)
6        12   FORMAT(1H1,'OUTPUT FROM IRAND'///)
7           KOUNTS=0
8           KOUNTF=0
9             DO 150 NTEST=1,N
0             READ(8,11) N,MS
1       11    FORMAT(I6,I12)
2             L=N-1
3             MI=(-1)/FLOAT(N-1)
4           S1=(1./FLOAT(N-1))*SQRT(N*(N-3.)/(N+1.))
5             MD=MI-2.*S1
6             SD=MI+2.*S1                                         .C33
7             CALL PINAX(MS)
8             DO 2 I=1,N
9       2     U(I)=IRAND(MS)/2.**35
0       C WE FIND THE PRODUCTS U(I)*U(I+1) FOR I=1,.....,N
1             I=1
2             SUM=0.
3       4     PROD=U(I)*U(I+1)
4             SUM=SUM+PROD
5             I=I+1
6             IF(I.GT.L) GO TO 3
7             GO TO 4
8       3     PROD=U(I)*U(1)
9             SUM=SUM+PROD
0             PROD1=N*SUM
1       C THE SQUARES OF THE SUM U(1)+U(2)+....+U(N) IS FOUND
2             SUM=0.
3             DO 5 I=1,N
4       5     SUM=SUM+U(I)
5             SUMSQ=SUM**2
6             NUMER=PROD1-SUMSQ
7             SUM=0.
8       C CALCULATION OF THE SUM OF THE SQUARES OF THE NUMBERS U(I)
9             DO 6 I=1,N
0             U(I)=U(I)**2
1       6     SUM=SUM+U(I)
2             PROD2=N*SUM
3             DENOM=PROD2-SUMSQ
4       C CALCULATION OF THE SERIAL CORRELATION COEFFICIENT.
5             C=NUMER/DENOM
6             IF(C.GE.MD.AND.C.LE.SD) GO TO 7
7             KOUNTF=KOUNTF+1
8             GO TO 150
9       7     KOUNTS=KOUNTS+1
0       150   CONTINUE
1       C * PS * IS THE PROBABILITY THE COR. COEFFICIENT TO BE IN T
2       C (MD,SD).
3             PS=KOUNTS/FLOAT(N)
4       C * PF * IS THE PROBABILITY THE COR. COEFFICIENT TO BE OUTS
5       C RANGE (MD,SD).
6             PF=KOUNTF/FLOAT(N)
7             WRITE(5,43) N,PS,PF
8         43  FORMAT(1H1,30X,'NUMBER OF TESTS PREFORMED ',I4//5X,'P
9           1E SERIAL COR. COEF. TO LIE IN THE RANGE (MD,SD) IS',F
0           2BILITY THE SERIAL COR. COEF. TO LIE OUTSIDE THE RANGE
1           3F4.3)
2             END
```

```
.0,GAUSS1
      C THIS SUBROUTINE SOLVES A SYSTEM OF LINEAR EQUATIONS USING THE
      C GAUSS-SEIDEL METHOD.
      C*****************************************************************/
      C A IS AN ARRAY HOLDING THE COEFFICIENTS OF THE EQUATIONS (INPUT)
      C B IS THE ARRAY OF THE CONSTANT TERMS OF THE EQUATIONS.(INPUT)
      C N IS THE SIZE OF THE ARRAY A.(INPUT)
      C EPS IS THE CONVERGENCE CRITERION.(INPUT)
      C MAXIM IS THE MAXIMUM NUMBER OF ITERATIONS DESIRED.(INPUT)
      C X IS THE SOLUTION VECTOR.(OUTPUT)
      C ITER IS THE MAXIMUM NUMBER OF THE ITRATIONS AT THE END
      C OF THE COMPUTATION.(OUTPUT)
            SUBROUTINE GAUSS(A,B,N,EPS,MAXIM,X,ITER)
            DIMENSION A(N,N),B(N),X(N)
            DO 10 L=1,N
      10    X(L)=0.
      C WE BEGIN THE ITERATION SCHEME.
            ITER=1
      99    BIG=0.
      C INDEX I SELECTS A ROW.
            DO 100 I=1,N
           SUM=0.
      C SEGMENT FROM HERE THROUGH STATEMENT 107 GETS THE SUM OF
      C THE TERMS IN A ROW, EXCLUDING THE MAIN DIAGONAL TERM.
            IF(I.EQ.1) GO TO 105
            LAST=I-1
            DO 106 J=1,LAST
      106   SUM=SUM+A(I,J)*X(J)
            IF(I.EQ.N) GO TO 103
      105   INITL=I+1
            DO 107 J=INITL,N
      107   SUM=SUM+A(I,J)*X(J)
      C COMPUTE NEW VALUE OF AN VARIABLE.
      103   TEMP=(B(I)-SUM)/A(I,I)
      C AT END OF SWEEP, THIS STATEMENT HAS PUT LARGEST RESIDUAL IN BIG.
            IF(ABS(TEMP-X(I)).GT.BIG) BIG=ABS(TEMP-X(I))
      100   X(I)=TEMP
      C IF LARGEST RESIDUAL IS LESS THAN EPSILON, PROCESS HAS CONVERGED.
            IF(BIG.LT.EPS) GO TO 85
      C IF ITERATION COUNTER EXCEEDS MAXIMUM , GIVE UP.
           IF(ITER.GE.MAXIM) GO TO 85
            ITER=ITER+1
            GO TO 99
      85    CONTINUE
            RETURN
            END
```

C5D

RANDSU12 1 010254 010312 0 044426 044632

SYS*RLIB$. LEVEL 68-2
END OF COLLECTION - TIME 3.545 SECONDS

@XQT PAO.MEET
PROBABILITY THE GAME TO BE WON BY A= .6334
PROBABILITY THE GAME TO BE WON BY B= .3666

1 010131 010246 2 044605 044611

IB$. LEVEL 68-2
COLLECTION - TIME 3.481 SECONDS

O.CESARO
THE NUMBER OF PAIRS WHICH ARE RELATIVELY PRIME IN TH E SAMPLE IS 12067

PROBABILITY THAT GCD(U,V)=1 IS .6033

S$*RLIB$. LEVEL 68-2
ND OF COLLECTION - TIME 3.077 SECONDS

QT PAO.MAIN1
NUMBER OF ITERATIONS PERFORMED 20

SOLUTION OF THE SYSTEM

X(1)= .07143
X(2)= .09821
X(3)= .07143
X(4)= .18750
X(5)= .25000
X(6)= .18750
X(7)= .42857
X(8)= .52679
X(9)= .42857

e51

```
-FO*PAO.WALK
      C    A RANDOM WALK  PROBLEM.
           DIMENSION P(3,3)
           KK=2**33
           MS=2228794509
          CALL PINAX(MS)
      C  THESE DOS CONTROL THE INITIAL POINT.
           DO 9 INITI=1,3
           DO 9 INITJ=1,3
      C  INITIALIZE FOR A RANDOM WALK.
           NUMSUC=0
      C  THIS DO PROVIDES FOR 30000 WALKS.
           DO 7 K=1,30000
           I=INITI
           J=INITJ
      C  GENERATING 1,2,3,4 AT RANDOM
    1      NUM=IRAND(MS)
           INDEX=(NUM/KK)+1
      C TAKE APPROPRIATE STEP, SEE IF WALK IS OVER.
           GO TO (2,3,4,5),INDEX
    2      I=I-1
           IF(I) 1,7,1
    3      J=J+1
           IF(J-4) 1,7,1
    4      I=I+1
           IF(I-4) 1,6,1
    5      J=J-1
           IF(J) 1,7,1
    6      NUMSUC=NUMSUC+1
    7      CONTINUE
           P(INITI,INITJ)=FLOAT(NUMSUC)/FLOAT(30000)
           WRITE(5,8) INITI,INITJ,P(INITI,INITJ),NUMSUC
    8      FORMAT(////5X,'PROBABILITY P(',I2,',',I2,')=',F6.4,5X,'NUMBER
          1 OF SUCCESSES',2X,I12/)
    9      CONTINUE
           END
```

PROBABILITY P(1, 1)= .0715 NUMBER OF SUCCESSES 2144

PROBABILITY P(1, 2)= .0974 NUMBER OF SUCCESSES 2921

PROBABILITY P(1, 3)= .0700 NUMBER OF SUCCESSES 2100

PROBABILITY P(2, 1)= .1870 NUMBER OF SUCCESSES 5611

PROBABILITY P(2, 2)= .2491 NUMBER OF SUCCESSES 7472

PROBABILITY P(2, 3)= .1848 NUMBER OF SUCCESSES 5545

PROBABILITY P(3, 1)= .4299 NUMBER OF SUCCESSES 12896

PROBABILITY P(3, 2)= .5299 NUMBER OF SUCCESSES 15898

PROBABILITY P(3, 3)= .4262 NUMBER OF SUCCESSES 12786

```
PA0.PRIMEPROG
      C THIS PROGRAM FINDS APPROXIMATELY THE PROBABILITY THAT A RANDOM NUMBER
      C IN THE RANGE 0,2**35-1 BE PRIME,AND ALSO THE APPROXIMATE NUMBER OF
      C PRIMES IN THE SAME RANGE.
      C PROGRAM READS THE NUMBER OF TESTS THAT WILL PERFORMED.
      C **SAMSIZ** IS THE SAMPLE SIZE.
            INTEGER SAMSIZ
            READ(8,108) NTEST,SAMSIZ
      108   FORMAT(I4,I6)
      C **SUMPP** IS THE SUM OF EACH PROBABILITY IN EACH TEST.
            SUMPP=0.
      C THE ITERATION SCHEME BEGINS.
            DO 48 J=1,NTEST
      C COMPUTER READS AN INTEGER NUMBER **MS** WHICH IS THE SEED FOR THE
      C GENERATOR IRAND.
            READ(8,20) MS
      20    FORMAT(I12)
      C **ISUMP** DENOTES THE NUMBER OF PAIRS,WHICH ARE IN THIS RANDOM SAMPLE.
            ISUMP=0
            CALL PINAX(MS)
            I=1
      C WE GENERATE A RANDOM NUMBER,IN THE RANGE 0,2**35-1
      1     N=IRAND(MS)
      C EXAMINE NOW IF THIS NO. IS PRIME.THE FUNCTION IPRIME WILL DECIDE THIS,
      C IF THE NO. IS PRIME THE OUTPUT VALUE OF THE FUNCTION WILL BE 1,
      C OTHERWISE 0.
            K=IPRIME(N)
            IF(K.EQ.1) GO TO 2
      3     I=I+1
            IF(I.GT.SAMSIZ) GO TO 4
            GO TO 1
      C WE ICREASE NOW THE NOS. OF PRIMES BY ONE.
      2     ISUMP=ISUMP+1
            GO TO 3
      C FIND THE PROBABILITY OF A NUMBER TO BE PRIME IN THIS RANDOM SAMPLE.
      4     PP=ISUMP/FLOAT(SAMSIZ)
      C WE INCREASE THE SUMPP BY PP.
            SUMPP=SUMPP+PP
            WRITE(5,10) J,ISUMP,PP
      10    FORMAT(1X,'NO OF PRIMES IN ',I2,'SAMPLE',2X,I10,3X,'PROBABILITY',
           13X,F10.3/)
      48    CONTINUE
      C **POPPR** DENOTES THE PROBABILITY OF THE POPULATION. POPULATION IS
      C ALL THE NUMBERS BETWEEN 0,2**35-1,
            POPPR=SUMPP/FLOAT(NTEST)
            NUMP=POPPR*(2.**35-1)
            WRITE(5,11) POPPR
      11    FORMAT(1X,'POPULATION_PROBABILITY IS',3X,F10,3/)
            WRITE(5,12) NUMP
      12    FORMAT(1X,'NUMBER OF PRIMES BETWEEN 0 AND 2**35-1 IS',3X,I12)
            END
```

```
@XQT PAO.PRIMEPROG
 NO OF PRIMES IN  1SAMPLE          35   PROBABILITY          .035
 NO OF PRIMES IN  2SAMPLE          46   PROBABILITY          .046
 NO OF PRIMES IN  3SAMPLE          52   PROBABILITY          .052
 NO OF PRIMES IN  4SAMPLE          47   PROBABILITY          .047
 NO OF PRIMES IN  5SAMPLE          43   PROBABILITY          .043
 NO OF PRIMES IN  6SAMPLE          44   PROBABILITY          .044
 NO OF PRIMES IN  7SAMPLE          41   PROBABILITY          .041
 NO OF PRIMES IN  8SAMPLE          40   PROBABILITY          .040
 NO OF PRIMES IN  9SAMPLE          55   PROBABILITY          .055
 NO OF PRIMES IN 10SAMPLE          42   PROBABILITY          .042
 NO OF PRIMES IN 11SAMPLE          50   PROBABILITY          .050
 NO OF PRIMES IN 12SAMPLE          52   PROBABILITY          .052
 NO OF PRIMES IN 13SAMPLE          44   PROBABILITY          .044
 NO OF PRIMES IN 14SAMPLE          35   PROBABILITY          .035
 NO OF PRIMES IN 15SAMPLE          35   PROBABILITY          .035
 NO OF PRIMES IN 16SAMPLE          50   PROBABILITY          .050
 NO OF PRIMES IN 17SAMPLE          52   PROBABILITY          .052
 NO OF PRIMES IN 18SAMPLE          50   PROBABILITY          .050
 NO OF PRIMES IN 19SAMPLE          39   PROBABILITY          .039
 NO OF PRIMES IN 20SAMPLE          45   PROBABILITY          .045
 POPULATION PROBABILITY IS :       .045


   NUMBER OF PRIMES BETWEEN 0 AND 2**35-1 IS      1541034160
 NORMAL EXIT.  EXECUTION TIME:       2013943 MILLISECONDS.

 @FIN
```

C5H

```
GOLF:> AO.IPRIME1
      1       C FUNCTION IPRIME DETERMINES IF AN INTEGER N IS PRIME OR NOT.
      2       C**********************************************************************
      3       C N IS AN INTEGER (INPUT)
      4       C THE RETURNED VALUE IS 1 IF N IS PRIME 0 OTHERWISE.
      5             FUNCTION IPRIME(N)
      6             IF(N.EQ.1.OR.N.EQ.2.OR.N.EQ.3.OR.N.EQ.5) GO TO 1
      7             RN=SQRT(FLOAT(N))
      8             M=RN+1
      9             GO TO 2
     10       1     IPRIME=1
     11             RETURN
     12       2     I=2
     13             IF(((N/I)*I-N).NE.0) GO TO 3
     14             IPRIME=0
     15             RETURN
     16       3     I=3
     17             IF(((N/I)*I-N).NE.0) GO TO 4
     18             IPRIME=0
     19             RETURN
     20       4     I=5
     21             IF(((N/I)*I-N).NE.0) GO TO 5
     22             IPRIME=0
     23             RETURN
     24       5     I=I+2
     25             IF(I.LT.M) GO TO 6
     26             IPRIME=1
     27             RETURN
     28       6     IF(((N/I)*I-N).NE.0) GO TO 7
     29             IPRIME=0
     30             RETURN
     31       7     I=I+4
     32             IF(I.LT.M) GO TO 8
     33             IPRIME=1
     34             RETURN
     35       8     IF(((N/I)*I-N).EQ.0) GO TO 9
     36             GO TO 5
     37       9     IPRIME=0
     38             RETURN
     39             END


$BRKPT PRINT$
```

```
A0.CESARO
      C THIS PROGRAM FINDS AN APPROXIMATE SOLUTION TO CESAROS THEOREM
            INTEGER U,V
      C INITIALIZATION
            ISUM1=0
            MS=6543808
            CALL PINAX(MS)
      C THIS DO EXAMINES 20000 PAIRS OF INTEGERS FINDING THEIR GCD.
            DO 1000 I=1,20000
      C WE CHOOSE TWO INTEGERS AT RANDOM
            U=IRAND(MS)
            V=IRAND(MS)
      C WE FIND THEIR GCD.
            K=IGCD(U,V)
      C WE CHECK IF THE FOUND GCD IS EQUAL TO 1
            IF(K.EQ.1) GO TO 2
      C NUMBER OF PAIRS OF INTEGERS WHICH HAVE AS GCD AN INTEGER DIFFE-
      C RENT FROM ONE.
            GO TO 1000
      C NUMBER OF PAIRS OF INTEGERS WHICH HAVE AS GCD THE UNITY.
       2    ISUM1=ISUM1+1
       1000 CONTINUE
            WRITE(5,11) ISUM1
       11   FORMAT(10X,'THE NUMBER OF PAIRS WHICH ARE RELATIVELY PRIME IN TH
           1E SAMPLE IS',2X,I5/)
      C WE FIND THE PROBABILITY THAT GCD IS EQUAL TO ONE.
            PSUC1=ISUM1/20000.
            WRITE(5,12) PSUC1
       12   FORMAT(10X,'PROBABILITY THAT GCD(U,V)=1 IS',2X,F10.4)
            END
```

```
A0.MULCHAPROG
            DIMENSION TOTAL(50)
            M=770
            N=2
            MS=34563209
            EX1=35.
            EX2=50.
            PRICCL=2.
            PRICME=5.
            CALL MULCHA(M,N,MS,EX1,EX2,PRICCL,PRICME,TCOST,TWT,TIDT,COSTC,
           1COSTM)
            TOTAL(N)=TCOST
            WRITE(5,10) M,N,TWT,TIDT,TOTAL(N)
       10   FORMAT(///10X,'TOTAL NUMBER OF ARRIVALS=',I3,10X,'NUMBER OF CLERKS
           1 USED=',I2/5X,'TOTAL WAITING TIME=',F6.3,1X,'HOURS',5X,'TOTAL IDLE
           2 TIME=',F6.3,1X,'HOURS',1X,'TOTAL COST=',F6.3)
            N1=N+1
       2    CALL MULCHA(M,N1,MS,EX1,EX2,PRICCL,PRICME,TCOST,TWT,TIDT,COSTC,
           1COSTM)
            TOTAL(N1)=TCOST
            WRITE(5,10) M,N1,TWT,TIDT,TOTAL(N1)
            IF(TOTAL(N1).LE.TOTAL(N)) GO TO 1
            GO TO 12
       1    N=N+1
            N1=N+1
            GO TO 2
       12   CONTINUE
            END
```

```
*PAO.GAMMA1
      C GENERATION OF GAMMA VARIATES.
      C*****************************************************************
      C RK IS A REAL NUMBER (INPUT).
      C A IS A REAL NUMBER. IT DENOTES THE PARAMETER A OF THE GAMMA
      C DISTRIBUTION.(INPUT).
      C Y IS THE ARRAY OF N RANDOM NUMBERS HAVING THE GAMMA
      C DISTRIBUTION (OUTPUT).
      C N IS THE NUMBER OF THE RANDOM NUMBERS DESIRED (INPUT).
      C MS IS THE STARTING VALUE (INTEGER) OF THE SUBROUTINE PINAX.
      C MS SHOULD BE CHOSEN IN THE INTERVAL (0,2**35-1).
      C DIFFERENT VALUES OF MS WILL RESULT IN DIFFERENT OUTPUT
      C SEQUENCES.AT THE END OF THE COMPUTATION MS CONTAINS THE
      C LAST NUMBER GENERATED BY IRAND(INPUT,OUTPUT).
            SUBROUTINE GAMMA(RK,A,Y,N,MS)
            DIMENSION Y(N),X(1)
           CALL PINAX(MS)
            ENTRY GAMMAF(RK,A,Y,N,MS)
            I=1
            U=IRAND(MS)/2.**35
            K=RK
            L=K+1
            P2=RK-K
            P1=1.-P2
    3       IF(U.LT.P1) GO TO 1
            CALL ERLANF(L,A,X,1,MS)
            Y(I)=X(1)
            GO TO 2
    1       CALL ERLANF(K,A,X,1,MS)
            Y(I)=X(1)
    2       I=I+1
            IF(I.GT.N) GO TO 4
            U=IRAND(MS)/2.**35
            GO TO 3
    4       CONTINUE
            RETURN
            END

*PAO.IGCD1
      C THIS FUNCTION CALCULATES THE GREATEST COMMON DIVISOR OF
      C TWO INTEGERS U AND V.
      C*****************************************************************
      C U IS AN INTEGER(INPUT)
      C V IS AN INTEGER(INPUT)
      C AT THE END OF THE COMPUTATION THE RETURNED VALUE WILL BE THE
      C GCD OF U,V. U AND V MAY BE POSITIVE OR NEGATIVE INTEGERS OR ZERO.
            FUNCTION IGCD(U,V)
            INTEGER U,V,R,TEMP
            IF(U.LT.0) U=-U
            IF(V.LT.0) V=-V
            IF(U.GT.V) GO TO 1
            TEMP=U
            U=V
            V=TEMP
     1      IF(V.EQ.0) GO TO 2
            R=MOD(U,V)
            U=V
            V=R
            GO TO 1
     2      IGCD=U
            RETURN
            END
```

```
PAO.INTEGRAL
      C THIS PROGRAM USES MONTE-CARLO TECHIQUES IN SOLVING A DETERMINISTIC
      C PROBLEM.
      C IT FINDS THE AREA OF THE FIRST QUANDRANT OF A UNIT CIRCLE.
      C **KOUNT** WILL EVENTUALLY CONTAIN THE NUMBER OF RANDOM PAIRS WHICH
      C WILL BE INSIDE THE CIRCLE.
            KOUNT =0
            N=30000
      C WE START THE ITERATION SCHEME. THE NUMBER OF ITERATIONS WILL BE 30000.
            I=1
      C WE DEFINE AN INITIAL VALUE FOR THE GENERATOR IRAND.
            MS=87211
            CALL PINAX(MS)
      C GENERATE A RANDOM PAIR.
      1     R1=IRAND(MS)/(2.**35)
            R2=IRAND(MS)/(2.**35)
            S=1.-R1**2
      C WE FIND THE FUNCTION G(R1)=SQRT(1.-R1**2)
            S=SQRT(S)
      C TEST IF THIS RANDOM PAIR IS INSIDE THE CIRCLE.
            IF(R2.LE.S) GO TO 2
      3     I=I+1
            IF(I.GT.N) GO TO 4
            GO TO 1
      C WE ICREASE THE NO. OF PAIRS WHICH ARE INSIDE THE CIRCLE BY ONE.
      2     KOUNT=KOUNT+1
            GO TO 3
      C NOW WE FIND THE AREA OF THE CIRCLE.
      4     CONTINUE
            AREA=FLOAT(KOUNT)/FLOAT(N)
            WRITE(5,12) AREA
      12    FORMAT(1X,'AREA OF THE FIRST QUAN. OF A UNIT CIRCLE IS',3X,F10.4)
            END
```

TOTAL NUMBER OF ARRIVALS=770 NUMBER OF CLERKS USED= 2
TOTAL WAITING TIME= 9.327 HOURS TOTAL IDLE TIME= 4.511 HOURS
TOTAL COST=55.659

TOTAL NUMBER OF ARRIVALS=770 NUMBER OF CLERKS USED= 3
TOTAL WAITING TIME= 1.323 HOURS TOTAL IDLE TIME=10.768 HOURS
TOTAL COST=28.152

TOTAL NUMBER OF ARRIVALS=770 NUMBER OF CLERKS USED= 4
TOTAL WAITING TIME= .243 HOURS TOTAL IDLE TIME=18.933 HOURS
TOTAL COST=39.479

SYS$*RLIB$. LEVEL 68-2
END OF COLLECTION - TIME 3.390 SECONDS

@XQT PAO.INTEGRAL
AREA OF THE FIRST QUAD. OF A UNIT CIRCLE IS .7806

```
GOLFO*PAO.GAME1
      C RECTANGULAR (TWO-PERSON) ZERO SUM GAME.
      C WITHOUT SADDLE POINTS.
      C SELECTING THE BEST STRATEGY FOR PLAYER R.
            DIMENSION IWIN(5)
            MS=1
            KK=2**33
            CALL PINAX(MS)
            LBESTC=3
      C THIS DO CONTROLS THE VARIOUS STRATEGIES .
            DO 1 J=1,5
      1     IWIN(J)=0
      C THIS DO CONTROLS 20000 PLAYS.
            DO 4 J=1,5
            LBESTR=J-1
            DO 4 I=1,20000
            NUMR=(IRAND(MS)/KK)+1
            NUMC=(IRAND(MS)/KK)+1
            IF(NUMR.LE.LBESTR) GO TO 3
            IF(NUMC.LE.LBESTC) GO TO 4
      2     IWIN(J)=IWIN(J)+1
            GO TO 4
      3     IF(NUMC.LE.LBESTC) GO TO 2
            IWIN(J)=IWIN(J)-2
      4     CONTINUE
      C WE FIND NOW WHICH STRATEGY WAS BEST.
            M=1
            DO 5 K=2,5
            IF(IWIN(M).GE.IWIN(K)) GO TO 5
            M=K
      5     CONTINUE
            DO 19 I=1,5
       19   WRITE(5,200) I,IWIN(I)
       200  FORMAT(10X,'IWIN(',I2,')=',I5)
      C SET LBESTR AT ITS BEST VALUE.
            LBESTR=M-1
            PP=LBESTR/4.
            WRITE(5,201) PP
       201  FORMAT(//'PROBABILITY TO SELECT ROW 1 SHOULD BE',1X,F4.2//)
           IWON=0
           WRITE(5,6)
      6    FORMAT(1X,'I AM READY TO PLAY')
       7     NUMR=(IRAND(MS)/KK)+1
           WRITE(5,8)
      8    FORMAT(1X,'YOUR TURN')
           READ(8,9,END=440) NUMCOL
      9    FORMAT(I2)
           IF(NUMR.LE.LBESTR) GO TO 15
           IF(NUMCOL.EQ.1) GO TO 13
           IWON=IWON+1
           WRITE(5,10)
      10   FORMAT(1X,'I CHOSE ROW 2 YOU CHOSE COL. 2 SO YOU PAY 1')
      11   WRITE(5,12) IWON
      12   FORMAT(1X,'I HAVE NOW WON',I10)
           GO TO 7
      13   WRITE(5,14)
      14   FORMAT(1X,'I CHOSE ROW 2 YOU CHOSE COL. 1 SO NO PAYMENT')
```

```
      GO TO 7
15    IF(NUMCOL.EQ.1) GO TO 17
      IWON=IWON-2
      WRITE(5,16)
16     FORMAT(1X,'I CHOSE ROW 1 YOU CHOSE COL 2 SO I PAY 2')
      GO TO 11
17    IWON=IWON+1
      WRITE(5,18)
18     FORMAT(1X,'I CHOSE ROW 1 YOU CHOSE COL 1 SO YOU PAY 1')
      GO TO 11
440   END
```

CONTINUED>

```
@XQT PAO.GAME1
              IWIN( 1)= 4981
              IWIN( 2)= 5016
              IWIN( 3)= 4903
              IWIN( 4)= 4756
              IWIN( 5)= 4952

 PROBABILITY TO SELECT ROW 1 SHOULD BE  .25

 I AM READY TO PLAY
 YOUR TURN
 I CHOSE ROW 1 YOU CHOSE COL 1 SO YOU PAY 1
 I HAVE NOW WON          1
 YOUR TURN
 I CHOSE ROW 1 YOU CHOSE COL 1 SO YOU PAY 1
 I HAVE NOW WON          2
 YOUR TURN
 I CHOSE ROW 2 YOU CHOSE COL. 2 SO YOU PAY 1
 I HAVE NOW WON          3
 YOUR TURN
 I CHOSE ROW 2 YOU CHOSE COL. 2 SO YOU PAY 1
 I HAVE NOW WON          4
 YOUR TURN
 I CHOSE ROW 1 YOU CHOSE COL 2 SO I PAY 2
 I HAVE NOW WON          2
 YOUR TURN
 I CHOSE ROW 2 YOU CHOSE COL. 2 SO YOU PAY 1
 I HAVE NOW WON          3
 YOUR TURN
 I CHOSE ROW 2 YOU CHOSE COL. 1 SO NO PAYMENT
 YOUR TURN
 I CHOSE ROW 2 YOU CHOSE COL. 2 SO YOU PAY 1
 I HAVE NOW WON          4
 YOUR TURN
 I CHOSE ROW 2 YOU CHOSE COL. 2 SO YOU PAY 1
 I HAVE NOW WON          5
 YOUR TURN
 I CHOSE ROW 2 YOU CHOSE COL. 2 SO YOU PAY 1
 I HAVE NOW WON          6
 YOUR TURN
NORMAL EXIT.  EXECUTION TIME:        16389 MILLISECONDS.


@FIN
```

```
*PAO.SAMPLE)
1       C SUBROUTINE SAMPLE SELECTS L RECORDS AT RANDOM FROM A SET OF N WHERE
2       C L GREATER THAN ZERO AND LESS THAN OR EQUAL TO N.
3       C*****************************************************************
4       C X IS AN INTEGER ARRAY CONTAINING N RECORDS (INPUT).
5       C Y IS AN INTEGER ARRAY(THE SAMPLE) WHICH CONTAINS L RECORDS
6       C SELECTED AT RANDOM FROM THE FILE WHICH CONTAINS N RECORDS(OUTPUT)
7       C N IS THE SIZE OF THE ARRAY X.(INPUT)
8       C L IS THE SIZE OF THE SAMPLE Y (INPUT)
9       C MS IS AN INTEGER WHICH IS USED AS AN INITIAL NUMBER FOR THE FUNCTION
0       C IRAND WHICH IS CONTAINED IN THE SUBROUTINE SAMPLE(INPUT)
1       C FOR DIFFERENT VALUES OF MS WE WILL HAVE DIFFERENT OUTPUT SAMPLES.
2             SUBROUTINE SAMPLE(X,Y,N,L,MS)
3              INTEGER X,Y,T
4              DIMENSION X(N),Y(L)
5              CALL PINAX(MS)
6              ENTRY QUICKS(X,Y,N,L,MS)
7              T=0
8              M=0
9        1     U=IRAND(MS)/(2.**35)
0              IF((N-T)*U.GE.(L-M)) GO TO 2
1       C SELECT A RECORD.
2              Y(M+1)=X(T+1)
3              M=M+1
4              T=T+1
5              IF(M.EQ.L) GO TO 3
6              GO TO 1
7        2     T=T+1
8              GO TO 1
9        3     CONTINUE
0              RETURN
1              END
```

```
*PAO.SHUFFL)
1       C SUBROUTINE SHUFFL SHUFFLES A SET OF N NUMBERS.
2       C*****************************************************************
3       C X IS AN INTEGER ARRAY OF N NUMBERS TO BE SHUFFLED.(INPUT,OUTPUT)
4       C N IS THE SIZE OF THE INPUT(OUTPUT) ARRAY X.(INPUT)
5       C MS IS AN INTEGER USED AS AN INITIAL NUMBER TO THE FUNCTION IRAND
6       C WHICH IS CONTAINED IN TJE SUBROUTINE SHUFFL.(INPUT)
7       C DIFFERENT VALUES OF MS WILL RESULT IN DIFFERENT VALUES OF THE
8       C INPUT SEQUENCE.
9             SUBROUTINE SHUFFL(X,N,MS)
0              INTEGER X,TEMP
1              DIMENSION X(N)
2              CALL PINAX(MS)
3              ENTRY PERMUT(X,N,MS)
4              J=N
5        1     U=IRAND(MS)/(2.**35)
6              K=J*U+1.
7       C EXCHANGE X(K) AND X(J).
8              TEMP=X(K)
9              X(K)=X(J)
0              X(J)=TEMP
1              J=J-1
2              IF(J.GT.1) GO TO 1
3              RETURN
               END
```

```
LFO*PAO.STAT
      C A STATISTICAL PROBLEM.
            INTEGER X(100),Y(6)
           REAL MEAN(60),H(5),A(6)
           WRITE(5,20)
       20  FORMAT(10X,'SAMPLE  NO.  DRAWN',10X,'CORRESPONDING HEIGHTS',
           110X,' MEAN  HEIGHT'//)
            READ(8,10) N,L,MS
       10   FORMAT(2I5,I12)
            DO 1 I=1,N
        1   X(I)=I-1
      C READ THE HEIGHTS OF THE 100 STUDENTS.
           READ(8,48) (H(K),K=1,5)
      48   FORMAT(5F4.1)
           SUM=0.
           SDEV=0.
           SUM1=0.
           KOUNT=1
      C A SAMPLE IS CHOSEN AT RANDOM.
           CALL SAMPLE(X,Y,N,L,MS)
      C WE FIND THE CORRESPONDING HEIGHT OF THE SAMPLE NUMBER OF A STUDENT
      C DRAWN FROM THE POPULATION OF 100 STUDENTS.
       21  J=1
      16   IF(Y(J).LE.4) GO TO 11
           IF(Y(J).LE.22) GO TO 12
           IF(Y(J).LE.64) GO TO 13
           IF(Y(J).LE.91) GO TO 14
           K=5
           GO TO 15
       11  K=1
           GO TO 15
       12  K=2
           GO TO 15
       13  K=3
           GO TO 15
       14  K=4
       15  A(J)=H(K)
           SUM1=SUM1+A(J)
           J=J+1
           IF(J.GT.L) GO TO 17
           GO TO 16
      C * MEAN * DENOTES THE MEAN OF THE SAMPLE.
       17  MEAN(KOUNT)=SUM1/FLOAT(J-1)
           WRITE(5,18) (Y(J),J=1,L),(A(J),J=1,L),MEAN(KOUNT)
      18   FORMAT(10X,6I3,10X,6F5.1,10X,F5.2)
           KOUNT=KOUNT+1
           SUM1=0.
           IF(KOUNT.GT.60) GO TO 22
      C ANOTHER SAMPLE IS DRAWN.
            CALL QUICKS(X,Y,N,L,MS)
           GO TO 21
       22  CONTINUE
           DO 23 I=1,60
       23  SUM=SUM+MEAN(I)
      C * SUM * DENOTES THE MEAN OF THE SAMPLING DISTRIBUTION OF MEANS.
           SUM=SUM/60.
           WRITE(5,24)
```

```
 24   FORMAT(40X,'MEAN OF THE SAMPLING DISTRIBUTION OF MEANS'/)
      WRITE(5,25) SUM
 25   FORMAT(60X,F6.2//)
      DO 26 KOUNT=1,60
      SDEV1=MEAN(KOUNT)-SUM
      SDEV1=SDEV1**2
      SDEV=SDEV+SDEV1
 26   CONTINUE
      SDEV=SDEV/60.
C SDEV IS THE STANDARD DEVIATION OF THE SAMPLING DISTRIB. OF MEANS.
      SDEV=SQRT(SDEV)
      WRITE(5,27)
 27   FORMAT(40X,'ST. DEVIATION OF THE SAMPLINGS DISTRIBUTIONS OF
     1MEANS'/)
      WRITE(5,28) SDEV
 28   FORMAT(65X,F6.2)
      END
```

CONTINUED

W.STAT

SAMPLE NO.	DRAWN					CORRESPONDING HEIGHTS						MEAN HEIGHT
20	30	42	56	84	91	64.0	67.0	67.0	67.0	70.0	70.0	67.50
31	37	49	63	76	93	67.0	67.0	67.0	67.0	70.0	70.0	68.00
3	13	15	79	89	94	61.0	64.0	64.0	70.0	70.0	73.0	67.00
3	33	61	73	77	87	61.0	67.0	67.0	70.0	70.0	70.0	67.50
2	62	78	83	86	90	61.0	67.0	70.0	70.0	70.0	70.0	68.00
12	32	36	49	67	75	64.0	67.0	67.0	67.0	70.0	70.0	67.50
6	47	63	69	91	94	64.0	67.0	67.0	67.0	70.0	73.0	68.00
12	26	36	54	69	75	64.0	64.0	67.0	67.0	67.0	70.0	66.50
14	23	37	70	97	98	64.0	67.0	67.0	70.0	73.0	73.0	69.00
14	15	21	26	35	77	64.0	64.0	64.0	67.0	67.0	70.0	66.00
3	16	24	25	40	79	61.0	64.0	67.0	67.0	67.0	70.0	66.00
3	53	70	93	92	94	61.0	67.0	70.0	70.0	73.0	73.0	69.00
45	52	61	72	79	95	67.0	67.0	67.0	70.0	70.0	73.0	69.00
15	22	33	64	73	77	64.0	64.0	67.0	67.0	70.0	70.0	67.00
18	26	53	72	84	93	64.0	67.0	67.0	70.0	70.0	73.0	68.50
0	21	32	70	77	81	61.0	64.0	67.0	70.0	70.0	70.0	67.00
14	21	25	38	55	79	64.0	64.0	67.0	67.0	67.0	70.0	66.50
24	25	55	73	84	89	67.0	67.0	67.0	70.0	70.0	70.0	68.50
17	31	56	67	89	96	64.0	67.0	67.0	70.0	70.0	73.0	68.50
18	59	63	67	70	88	64.0	67.0	67.0	70.0	70.0	70.0	68.00
45	66	71	83	86	89	67.0	70.0	70.0	70.0	70.0	70.0	69.50
14	39	46	51	57	85	64.0	67.0	67.0	67.0	67.0	70.0	67.00
19	33	58	59	62	83	64.0	67.0	67.0	67.0	67.0	70.0	67.00
30	51	57	58	68	88	67.0	67.0	67.0	67.0	70.0	70.0	68.00
1	17	24	33	54	65	61.0	64.0	67.0	67.0	67.0	70.0	66.00
22	23	33	40	56	73	64.0	67.0	67.0	67.0	67.0	70.0	67.00
2	15	16	34	48	51	61.0	64.0	64.0	67.0	67.0	67.0	65.00
9	26	29	76	81	94	64.0	67.0	67.0	70.0	70.0	73.0	68.50
3	4	22	64	85	88	61.0	61.0	64.0	70.0	70.0	70.0	66.00
31	39	72	82	85	89	67.0	67.0	70.0	70.0	70.0	70.0	69.00
3	22	33	50	82	86	61.0	64.0	67.0	67.0	70.0	70.0	66.50
15	36	51	63	89	91	64.0	67.0	67.0	67.0	70.0	70.0	67.50
5	15	46	63	89	91	64.0	64.0	67.0	67.0	70.0	70.0	67.00
21	23	33	61	95	98	64.0	67.0	67.0	67.0	73.0	73.0	68.50
49	64	77	79	93	98	67.0	67.0	70.0	70.0	73.0	73.0	70.00
17	31	34	38	61	62	64.0	67.0	67.0	67.0	67.0	67.0	66.50
1	19	25	48	56	86	61.0	64.0	67.0	67.0	67.0	70.0	66.00
34	48	70	77	80	92	67.0	67.0	70.0	70.0	70.0	73.0	69.50
8	19	39	56	66	78	64.0	64.0	67.0	67.0	70.0	70.0	67.00

3	23	34	36	66	78	61.0	67.0	67.0	67.0	70.0	70.0	67.00
2	19	20	37	50	51	61.0	64.0	64.0	67.0	67.0	67.0	65.00
6	69	79	80	93	98	64.0	70.0	70.0	70.0	73.0	73.0	70.00
0	9	19	32	33	85	61.0	64.0	64.0	67.0	67.0	70.0	65.50
16	18	25	26	43	75	64.0	64.0	67.0	67.0	67.0	70.0	66.50
45	63	66	77	84	88	67.0	67.0	70.0	70.0	70.0	70.0	69.00
2	23	55	71	73	80	61.0	67.0	67.0	70.0	70.0	70.0	67.50
4	5	18	61	75	86	61.0	64.0	64.0	67.0	70.0	70.0	66.00
18	22	32	34	39	54	64.0	64.0	67.0	67.0	67.0	67.0	66.00
3	5	11	68	76	91	61.0	64.0	64.0	70.0	70.0	70.0	66.50
32	39	48	50	60	97	67.0	67.0	67.0	67.0	67.0	73.0	68.00
16	18	25	34	50	63	64.0	64.0	67.0	67.0	67.0	67.0	66.00
27	33	34	39	69	84	67.0	67.0	67.0	67.0	70.0	70.0	68.00
33	48	53	57	71	74	67.0	67.0	67.0	67.0	70.0	70.0	68.00
0	15	28	44	52	69	61.0	64.0	67.0	67.0	67.0	70.0	66.00
21	25	33	57	67	96	64.0	67.0	67.0	67.0	70.0	73.0	68.00
28	35	42	51	57	73	67.0	67.0	67.0	67.0	67.0	70.0	67.50
3	13	65	70	77	88	61.0	64.0	70.0	70.0	70.0	70.0	67.50
7	16	66	69	70	97	64.0	64.0	70.0	70.0	70.0	73.0	68.50
11	29	30	44	58	70	64.0	67.0	67.0	67.0	67.0	70.0	67.00
7	39	44	60	64	96	64.0	67.0	67.0	67.0	67.0	73.0	67.50

MEAN OF THE SAMPLING DISTRIBUTION OF MEANS

67.42

ST. DEVIATION OF THE SAMPLINGS DISTRIBUTIONS OF MEANS

1.19

CONTINUED

```
A0.GAMMA1
      C GENERATION OF GAMMA VARIATES.
      C*************************************************************************
      C RK IS A REAL NUMBER (INPUT).
      C A IS A REAL NUMBER. IT DENOTES THE PARAMETER A OF THE GAMMA
      C DISTRIBUTION.(INPUT).
      C Y IS THE ARRAY OF N RANDOM NUMBERS HAVING THE GAMMA
      C DISTRIBUTION (OUTPUT).
      C N IS THE NUMBER OF THE RANDOM NUMBERS DESIRED (INPUT).
      C MS IS THE STARTING VALUE (INTEGER) OF THE SUBROUTINE PINAX.
      C MS SHOULD BE CHOSEN IN THE INTERVAL (0,2**35-1).
      C DIFFERENT VALUES OF MS WILL RESULT IN DIFFERENT OUTPUT
      C SEQUENCES.AT THE END OF THE COMPUTATION MS CONTAINS THE
      C LAST NUMBER GENERATED BY IRAND(INPUT,OUTPUT).
            SUBROUTINE GAMMA(RK,A,Y,N,MS)
            DIMENSION Y(N),X(1)
           CALL PINAX(MS)
            ENTRY GAMMAF(RK,A,Y,N,MS)
            I=1
            U=IRAND(MS)/2.**35
            K=RK
            L=K+1
            P2=RK-K
            P1=1.-P2
      3     IF(U.LT.P1) GO TO 1
            CALL ERLANF(L,A,X,1,MS)
            Y(I)=X(1)
            GO TO 2
      1     CALL ERLANF(K,A,X,1,MS)
            Y(I)=X(1)
      2     I=I+1
            IF(I.GT.N) GO TO 4
            U=IRAND(MS)/2.**35
            GO TO 3
      4     CONTINUE
            RETURN
            END


0,EXP1,,SAMPLE1,,SHUFFL1,,POISSN1,,HYPGEO1,,NBIN1,,PASCAL1
```

PAO.ERLANG1

```
      C GENERATION OF ERLANG VARIATES.
      C*************************************************************
      C K IS AN INTEGER(INPUT).
      C A IS A REAL NUMBER.THIS DENOTES THE PARAMETER A OF THE ERLANG
      C DISTRIBUTION.(INPUT).
      C X IS AN ARRAY OF N RANDOM NUMBERS HAVING THE ERLANG
      C DISTRIBUTION (OUTPUT)
      C N IS THE NUMBER OF THE RANDOM NUMBERS DESIRED (INPUT).
      C MS IS THE STARTING VALUE(INTEGER) OF THE SUBROUTINE PINAX.
      C MS SHOULD BE CHOSEN IN THE INTERVAL (0,2**35-1).
      C DIFFERENT VALUES OF MS WILL RESULT IN DIFFERENT OUTPUT
      C SEQUENCES. AT THE END OF THE COMPUTATION MS CONTAINS THE
      C LAST NUMBER GENERATED BY IRAND (INPUT,OUTPUT).
            SUBROUTINE ERLANG(K,A,X,N,MS)
            DIMENSION X(N)
            CALL PINAX(MS)
            ENTRY ERLANF(K,A,X,N,MS)
            DO 1 I=1,N
            RT=1.
            DO 2 L=1,K
            R=IRAND(MS)/2.**35
      2     RT=RT*R
            X(I)=-ALOG(RT)/A
      1     CONTINUE
            RETURN
            END
```

PAO.LOGNOR1

```
      C THIS SUBROUTINE GENERATES RANDOM NUMBERS LOGNORMALLY DISTRIBUTED.
      C *************************************************************
      C X IS AN ARRAY OF LOGNORMALLY DISTRIBUTED RANDOM NUMBERS (OUTPUT).
      C N IS THE SIZE OF THE ARRAY X (INPUT)
      C EX IS THE DESIRED MEAN OF THE RANDOM NUMBERS (INPUT)
      C STDX IS THE DESIRED ST. DEVIATION OF THE RANDOM NUMBERS.(INPUT)
      C MS IS THE STARTING VALUE(INTEGER) OF THE SUBROUTINE PINAX.
      C MS SHOULD BE CHOSEN IN THE INTERVAL (0,2**35-1).
      C DIFFERENT VALUES OF MS WILL RESULT IN DIFFERENT OUTPUT
      C SEQUENCES. AT THE END OF THE COMPUTATION MS CONTAINS THE
      C LAST NUMBER GENERATED BY IRAND(INPUT,OUTPUT)
            SUBROUTINE LOGNOR(X,N,EX,STDX,MS)
            REAL MEANSQ
            DIMENSION R(12)
            DIMENSION X(N)
            CALL PINAX(MS)
            ENTRY LOGNRF(X,N,EX,STDX,MS)
            MEANSQ=EX**2
            VX=STDX**2
            EY=ALOG(EX)-0.5*ALOG(STDX/MEANSQ+1)
            VY=ALOG(VX/MEANSQ+1)
            DO 1 I=1,N
            SUM=0.
            DO 2 J=1,12
            R(J)=IRAND(MS)/(2.**35)
      2     SUM=SUM+R(J)
            X(I)=EXP(EY+VY*(SUM-6.))
      1     CONTINUE
            RETURN
            END
```

```
)*PAO.PASCAL1
 1       C THIS SUBROUTINE GENERATES RANDOM GEOMETRIC VARIATES.
 2       C*****************************************************************
 3       C Q IS THE PROBABILITY OF FAILURE (INPUT)
 4       C IX IS AN ARRAY OF RANDOM NUMBERS HAVING THE GEOMETRIC
 5       C DISTRIBUTION (OUTPUT)
 6       C N IS THE SIZE OF THE ARRAY IX.
 7       C MS IS THE STARTING VALUE (INTEGER) OF THE SUBROUTINE PINAX.
 8       C MS SHOULD BE CHOSEN IN THE INTERVAL (0,2**35-1)
 9       C DIFFERENT VALUES OF MS RESULT IN DIFFERENT OUTPUT SEQUENCES.
10       C AT THE END OF THE COMPUTATION MS CONTAINS THE LAST NUMBER
11       C GENERATED BY IRAND(INPUT,OUTPUT)
12             SUBROUTINE PASCAL(Q,IX,N,MS)
13              DIMENSION IX(N)
14              CALL PINAX(MS)
15             ENTRY PASCQ(Q,IX,N,MS)
16              Q1=ALOG(Q)
17              DO 1 I=1,N
18              R=IRAND(MS)/2.**35
19              R=ALOG(R)
20              IX(I)=R/Q1
21              IX(I)=IX(I)+1
22        1     CONTINUE
23              RETURN
24              END
```

```
)*PAO.NBIN1
 1       C GENERATION OF NEGATIVE BINOMIAL VARIATES.
 2       C*****************************************************************
 3       C K DENOTES THE NUMBER OF SUCCESSES.(INPUT)
 4       C Q IS THE PROBABILITY OF FAILURE(INPUT)
 5       C IX IS AN ARRAY OF NEGATIVE BINOMIAL VARIATES.(OUTPUT)
 6       C N IS THE SIZE OF THE ARRAY IX.
 7       C MS IS THE STARTING VALUE (INTEGER) OF THE SUBROUTINE PINAX.
 8       C MS SHOULD BE CHOSEN IN THE INTERVAL (0,2**35-1)
 9       C DIFFERENT VALUES OF MS RESULT IN DIFFERENT OUTPUT SEQUENCES.
10       C AT THE END OF THE COMPUTATION MS CONTAINS THE LAST NUMBER
11       C GENERATED BY IRAND(INPUT,OUTPUT)
12             SUBROUTINE NBIN(K,Q,IX,N,MS)
13              DIMENSION IX(N)
14              CALL PINAX(MS)
15             ENTRY NBINF(K,Q,IX,N,MS)
16              Q1=ALOG(Q)
17              DO 1 I=1,N
18              T=1.
19              DO 6 J=1,K
20              R=IRAND(MS)/2.**35
21        6     T=T*R
22        1     IX(I)=ALOG(T)/Q1
23              RETURN
24              END
```

```
*PAO.HYPGEO1
      C GENERATION OF RANDOM NUMBERS HAVING THE HYPERGEOMETRIC DISTRIBUTION.
      C**********************************************************************
      C NP IS THE SIZE OF THE POPULATION (INPUT)
      C M IS THE SIZE OF THE SAMPLE.(INPUT)
      C P IS THE PROPORTION OF THE TOTAL POPULATION CONSISTING OF
      C CLASS-I ELEMENTS.(INPUT)
      C IX IS THE ARRAY OF N RANDOM NUMBERS FOLLOWING THE HYPERGEOMETRIC
      C DISTRIBUTION (OUTPUT)
      C N IS THE SIZE OF THE ARRAY IX.(INPUT)
      C MS IS THE STARTING VALUE (INTEGER) OF THE SUBROUTINE PINAX.
      C MS SHOULD BE CHOSEN IN THE INTERVAL (0,2**35-1)
      C DIFFERENT VALUES OF MS RESULT IN DIFFERENT OUTPUT SEQUENCES.
      C AT THE END OF THE COMPUTATION MS CONTAINS THE LAST NUMBER
      C GENERATED BY IRAND(INPUT,OUTPUT)
            SUBROUTINE HYPGEO(NP,M,P,IX,N,MS)
            DIMENSION IX(N)
            CALL PINAX(MS)
            ENTRY HYPGEF(NP,M,P,IX,N,MS)
            DO 1 I=1,N
            IX(I)=0
            NT=NP
            PT=P
            DO 2 J=1,M
            R=IRAND(MS)/2.**35
            IF(R-PT) 6,6,9
      6     S=1.
            IX(I)=IX(I)+1
            GO TO 10
      9     S=0.
      10    PT=(NT*PT-S)/(NT-1.)
      2     NT=NT-1
      1     CONTINUE
            RETURN
            END


*PAO.POISSN1
      C GENERATION OF POISSON VARIATES.
      C **********************************************************************
      C P IS THE DESIRED MEAN OF THE POISSON VARIATES.(INPUT)
      C IX IS AN ARRAY OF INTEGERS FOLLOWING THE POISSON
      C DISTRIBUTION.(OUTPUT)
      C N IS THE SIZE OF THE ARRAY IX.
      C MS IS THE STARTING VALUE (INTEGER) OF THE SUBROUTINE PINAX.
      C MS SHOULD BE CHOSEN IN THE INTERVAL (0,2**35-1)
      C DIFFERENT VALUES OF MS RESULT IN DIFFERENT OUTPUT SEQUENCES.
      C AT THE END OF THE COMPUTATION MS CONTAINS THE LAST NUMBER
      C GENERATED BY IRAND(INPUT,OUTPUT)
            SUBROUTINE POISSN(P,IX,N,MS)
            DIMENSION IX(N)
            CALL PINAX(MS)
            ENTRY POISSF(P,IX,N,MS)
            P1=EXP(-P)
            DO 1 I=1,N
            IX(I)=0
            Q=1.
      2     U=IRAND(MS)/2.**35
            Q=Q*U
            IF(Q.LT.P1) GO TO 1
            IX(I)=IX(I)+1
            GO TO 2
      1     CONTINUE
            RETURN
            END
```

```
*PAO.MODEL
1       C SINGLE CHANNEL QUEUING MODEL.
2             DIMENSION X(1),Y(1)
3             REAL IDT
4             MS=761209265
5       C SET INITIAL CONDITIONS.
6             SUMST=0.
7             IDT=0.
8             WT=0.
9             TWT=0.
0             TIDT=0.
1       C EXPECTED VALUE OF THE ARRIVAL TIME.
2             EX=50.
3       C EXPECTED VALUE OF THE SERVICING TIME.
4             EX1=35.
5             N=1
6             KOUNT=0
7       C GENERATION OF THE ARRIVAL TIME.
8             CALL EXPR(X,EX,N,MS)
9        1    AT=X(1)
0             AT=AT-WT
1       C GENERATION OF THE SERVICE TIME.
2             CALL EXPO(Y,EX1,N,MS)
3             ST=Y(1)
4             SUMST=SUMST+ST
5             IF(ST.GT.AT) GO TO 2
6             IF(ST.LT.AT) GO TO 3
7             WT=0.
8             IDT=0.
9        45   KOUNT=KOUNT+1
0             IF(KOUNT.EQ.10000) GO TO 50
1       C ANOTHER UNIT ENTERS THE SYSTEM FOR SERVICE.
2             CALL EXPO(X,EX,N,MS)
3             GO TO 1
4        2    IDT=0.
5             WT=ST-AT
6             TWT=TWT+WT
7             GO TO 45
8        3    WT=0.
9             IDT=AT-ST
0             TIDT=TIDT+IDT
1             GO TO 45
2       C CALCULATION OF THE EXPECTED WAITING TIME,IDLE TIME AND TIME
3       C SPENT IN THE SYSTEM BY A CUSTOMER.
4        50   EWT=TWT/10000.
5             EIDT=TIDT/10000.
6             SUMST=TWT+SUMST
7             ESUMST=SUMST/10000.
8             WRITE(5,10) EWT,ESUMST,EIDT
9        10   FORMAT(1H1,10X,'EXPECTED WAITING TIME=',F7.2//10X,'EXPECTED TIME
0           1 SPENT IN THE SYSTEM BY A CUSTOMER=',F7.2//10X,'EXPECTED IDLE TIME
1           2 =',F7.2)
2             END
```

```
*PAO.MULCHA1
      C THIS SUBROUTINE SIMULATES A MULTICHANNEL QUEUING MODEL.
      C*****************************************************************
      C M IS THE NUMBER OF TOTAL ARRIVALS.(INPUT)
      C N IS THE NUMBER OF CLERKS AT THE SERVICE STATIONS.(INPUT)
      C MS IS AN INTEGER .(INPUT). THIS NUMBER IS USED BY SUBROUTINE EXPR
      C WHICH IS CONTAINED IN THE SUB. MULCHA.(INPUT)
      C EX1 IS THE MEAN ARRIVAL TIME.(INPUT)
      C EX2 IS THE MEAN SERVICE TIME.(INPUT)
      C PRICCL IS THE LABOR COST OF A CLERK PER HOUR.(INPUT)
      C PRICME IS THE LABOR COST OF A CUSTOMER PER HOUR.(INPUT)
      C TCOST IS THE TOTAL COST ON THE PART OF CUSTOMERS AND CLERKS.(OUTPUT)
      C TWT IS THE TOTAL WAITING TIME IN HOURS.(OUTPUT)
      C TIDT IS THE TOTAL IDLE TIME IN HOURS.(OUTPUT)
      C COSTC IS THE TOTAL LABOR COST ON THE PART OF CLERKS.(OUTPUT)
      C COSTM IS THE TOTAL LABOR COST ON THE PART OF CUSTOMERS(MECHANICS)
      C (OUTPUT).
            SUBROUTINE MULCHA(M,N,MS,EX1,EX2,PRICCL,PRICME,TCOST,TWT,TIDT,
           1COSTC,COSTM)
            REAL IDT
            DIMENSION IDT(100),ST(1),WT(100),TT(100),AT(1)
      C INOTIALIZE.
            TAT=0.
            IDT(1)=0.
            TIDT=0.
            TWT=0.
      C GENERATION OF THE ARRIVAL TIME.
            CALL EXPR(AT,EX1,1,MS)
            J=2
      2     TAT=TAT+AT(1)
            IDT(J)=TAT
            TIDT=TIDT+IDT(J)
            J=J+1
            IF(J.GT.N) GO TO 3
      C ANOTHER UNIT ENTERS THE SYSTEM
            CALL EXPQ(AT,EX1,1,MS)
            GO TO 2
      3     I=N
            DO 4 J=1,N
      C GENERATION OF THE SERVICE TIME.
             CALL EXPQ(ST,EX2,1,MS)
            TT(J)=IDT(J)+ST(1)
      4     CONTINUE
      C THIS SUBROUTINE FINDS THE POSITION OF THE MINIMUM ELEMENT IN THE
      C ARRAY TT.
      100   CALL MINIM(TT,N,L)
            I=I+1
            IF(I.GE.M) GO TO 150
            CALL EXPQ(AT,EX1,1,MS)
            TAT=TAT+AT(1)
            DIF=TAT-TT(L)
            IF(DIF.GT.0.) GO TO 15
            IF(DIF.EQ.0.) GO TO 16
            WT(L)=-DIF
      C COMPUTATION OF THE TOTAL WAITING TIME.
            TWT=TWT+WT(L)
            IDT(L)=0
```

```
.TUBE1
      SUBROUTINE TUBE(N,TIME,EX,STDX,DOWND,DOWNN,P,PRICE,FDAY,FNIGHT,
     1COST,MS)
      DIMENSION W(1000),ST(1)
      INTEGER FDAY,FNIGHT
      COST=0.
      FDAY=0.
      FNIGHT=0
      CALL RANDNR(W,N,EX,STDX,MS)
      CALL SORTRL(N,W)
      K=1
    4 IF(W(K).GT.TIME) GO TO 1
   14 U=IRAND(MS)/2.**35
      IF(U.LT.P) GO TO 2
      FNIGHT=FNIGHT+1
      COST=COST+DOWNN+PRICE
      GO TO 3
    2 FDAY=FDAY+1
      COST=COST+DOWND+PRICE
    3 CALL FASTNR(ST,1,EX,STDX,MS)
      W(K)=W(K)+ST(1)
      K=K+1
      IF(K.GT.N) GO TO 1
      GO TO 4
    1 CALL SORTRL(N,W)
      IF(W(1).LE.TIME) GO TO 5
      GO TO 15
    5 K=1
      GO TO 14
   15 CONTINUE
      RETURN
      END
```

```
$RAO,TUBE,R06
      INTEGER FDAY,FNIGHT,SUM
C INPUT TO THE SUBROUTINE TUBE.
      N=100
      TIME=740.
      EX=160.
      STDX=14.
      DOWND=50.
      DOWNN=100.
      P=.7
      PRICE=5.
      MS=7659085
C CASE 1. REPLACE THE TUBES INDIVIDUALLY AS THEY FAIL.
      CALL TUBE(N,TIME,EX,STDX,DOWND,DOWNN,P,PRICE,FDAY,FNIGHT,COST,MS)
      SUM=FDAY+FNIGHT
      WRITE(5,10) TIME,SUM,FDAY,FNIGHT,COST
   10 FORMAT(10X,'IN A TIME PERIOD OF',1X,F5.0,1X,'DAYS',5X,'THE TUBES W
     1HICH FAILED WERE',1X,I4/10X,I5,1X,'TUBES FAILED DURING THE DAY AND
     2',1X,I5,1X,'FAILED AT NIGHT'/10X,'THE TOTAL COST OF REPLACING THE
     3TUBES WAS',1X,F8.1//)
C CASE 2.REPLACE ALL TUBES EVERY 5 MONTHS AND REPLACE THE TUBES THAT
C FAIL DURING THR INTERIM PERIOD INDIVIDUALLY.
      COUNT=150.
      COST1=0.
      TIME1=150.
    1 CALL TUBE(N,TIME1,EX,STDX,DOWND,DOWNN,P,PRICE,FDAY,FNIGHT,COST,MS)
      COST1=COST1+COST
      COST1=COST1+2.*N
      COUNT=COUNT+150.
      IF(COUNT.LE.TIME) GO TO 1
      COUNT=COUNT-150.
      COUNT=TIME-COUNT
      CALL TUBE(N,COUNT,EX,STDX,DOWND,DOWNN,P,PRICE,FDAY,FNIGHT,COST,MS)
      COST1=COST1+COST
      WRITE(5,11) TIME,COST1
   11 FORMAT(10X,'IN A TIME PERIOD OF',1X,F5.0,1X,'DAYS'/10X,'THE TOTA
     1L COST OF REPLACING THE TUBES WAS',1X,F8.1)
      END
```

```
GOLFO*.FAO.PASBIG)
      C GENERATION OF RANDOM GEOMETRIC VARIATES WHEN P (PROBABILITY OF
      C SUCCESS) IS LARGE.
      C*************************************************************
      C P IS THE PROBABILITY OF SUCCES(INPUT)
      C IX IS AN ARRAY OF RANDOM NUMBERS HAVING THE GEOMETRIC
      C DISTRIBUTION.(OUTPUT)
      C MS IS THE STARTING VALUE (INTEGER) OF THE SUBROUTINE PINAX.
      C IS SHOULD BE CHOSEN IN THE INTERVAL (0,2**35-1)
      C DIFFERENT VALUES OF MS RESULT IN DIFFERENT OUTPUT SEQUENCES.
      C AT THE END OF THE COMPUTATION MS CONTAINS THE LAST NUMBER
      C GENERATED BY IRAND(INPUT,OUTPUT)
            SUBROUTINE PASBIG(P,IX,N,MS)
            DIMENSION IX(N)
            CALL PINAX(MS)
            ENTRY PAFAST(P,IX,N,MS)
            DO 1 I=1,N
            K=1
      2     R=IRAND(MS)/2.**35
            IF(R.LE.P) GO TO 3
            K=K+1
            GO TO 2
      3     IX(I)=K
      1     CONTINUE
            RETURN
            END
GOLFO*.FAO.NBIN)
      C GENERATION OF NEGATIVE BINOMIAL VARIATES WITH PARAMETER RK (NUMBER
      C OF SUCCESSES) A REAL NUMBER.
      C*************************************************************
      C RK IS A REAL NUMBER.(INPUT)
      C Q IS THE PROBABILITY OF FAILURE.(INPUT)
      C IY IS AN ARRAY OF INTEGERS HAVING THE NEGATIVE BINOMIAL
      C DISTRIBUTION. (OUTPUT)
      C N IS THE SIZE OF THE ARRAY IY.(INPUT)
      C MS IS THE STARTING VALUE (INTEGER) OF THE SUBROUTINE PINAX.
      C MS SHOULD BE CHOSEN IN THE INTERVAL (0,2**35-1)
      C DIFFERENT VALUES OF MS RESULT IN DIFFERENT OUTPUT SEQUENCES.
      C AT THE END OF THE COMPUTATION MS CONTAINS THE LAST NUMBER
      C GENERATED BY IRAND(INPUT,OUTPUT)
            SUBROUTINE RBIN(RK,Q,IY,N,MS)
            DIMENSION IY(N),IX(1)
            CALL PINAX(MS)
            ENTRY RBINF(RK,Q,IY,N,MS)
            I=1
            U=IRAND(MS)/(2.**35)
            K=RK
            L=K+1
            P2=RK-K
            P1=1-P2
      3     IF(U.LT.P1) GO TO 1
            CALL NBINF(L,Q,IX,1,MS)
            IY(I)=IX(1)
            GO TO 2
      1     CALL NBINF(K,Q,IX,1,MS)
            IY(I)=IX(1)
      2     I=I+1
            IF(I.GT.N) GO TO 4
            U=IRAND(MS)/(2.**35)
```

INDEX OF PROGRAMS AND ROUTINE LISTINGS